Xiaofei Wang · Yiwen Han
Victor C. M. Leung · Dusit Niyato
Xueqiang Yan · Xu Chen

Edge AI

Convergence of Edge Computing and Artificial Intelligence

Springer

Edge AI

Xiaofei Wang • Yiwen Han • Victor C. M. Leung •
Dusit Niyato • Xueqiang Yan • Xu Chen

Edge AI

Convergence of Edge Computing and Artificial Intelligence

Springer

Xiaofei Wang
College of Intelligence and Computing
Tianjin University
Tianjin, Tianjin, China

Yiwen Han
College of Intelligence and Computing
Tianjin University
Tianjin, Tianjin, China

Victor C. M. Leung
College of Computer Science and Software Engineering
Shenzhen University
Shenzhen, Guangdong, China

Dusit Niyato
School of Computer and Engineering
Nanyang Technological University
Singapore, Singapore

Xueqiang Yan
2012 Lab
Huawei Technologies (China)
Shenzhen, China

Xu Chen
School of Data and Computer Science
Sun Yat-sen University
Guangzhou, Guangdong, China

ISBN 978-981-15-6185-6 ISBN 978-981-15-6186-3 (eBook)
https://doi.org/10.1007/978-981-15-6186-3

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020
This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Preface

At present, we are living in an era of rapid development. Artificial intelligence (AI), as a technology leading the trend of this era and subverting people's traditional lifestyle, is deeply integrated into production and life around the world. With its rapid rise in the fields of smart factories, smart cities, smart homes, and smart Internet of Things, it uses technology to achieve the interaction between humans and machines. At present, AI has replaced human roles in many key areas of high intensity, difficulty, and danger and even has surpassed human capabilities in some areas. Therefore, AI has promoted the liberation of human freedom to a certain extent by replacing labor and management of human beings.

However, the realization of AI requires large-scale data as support. It is based on the training and learning of a large number of sample data that AI can perform nearly or even surpass human performance. The current data in the network is showing an exponential growth, which has created opportunities for the rise and development of AI. However, the rapid increase in the size of network data is a great challenge of current network architecture. In order to alleviate the pressure on the network caused by the explosive growth of data scale, edge computing technology came into being. Edge computing can reduce the pressure on the network and reduce the delay in request response by setting distributed edge nodes at the edge of the network.

The explosive growth of data not only provides a prerequisite for the development of AI but also creates opportunities for the rise of edge computing. However, AI and edge computing, as two popular emerging technologies, are inextricably linked to each other. On the one hand, the characteristics of edge computing that can reduce latency and traffic load can provide basic guarantees for AI. On the other hand, the learning and decision-making capabilities of AI are supporting the efficient and stable operation of edge computing. The two technologies are not only mutually supporting but also merging with each other, and they are inseparable. Deep learning, as the most representative technology of AI in combination with edge computing, has made remarkable progress in many fields through cooperation with edge computing. In this background, the purpose of this monograph is to explore the

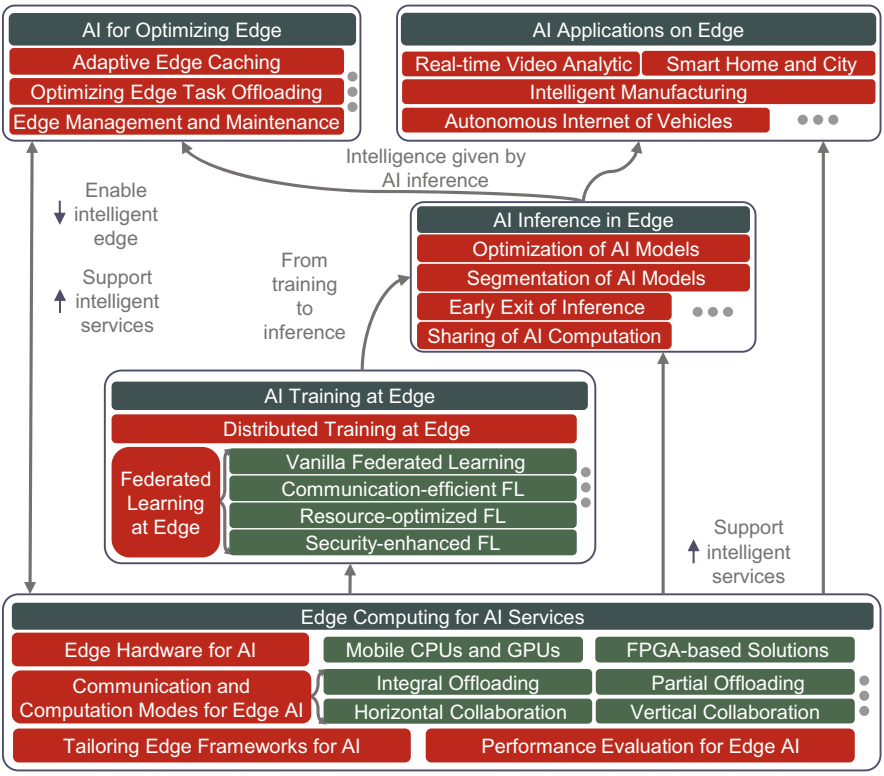

Fig. 1 Conceptual relationships of edge intelligence and intelligent edge

relevant achievements around the relationship between edge computing and artificial intelligence.

This monograph introduces and discusses the advanced technology of edge AI in terms of fundamentals, concepts, frameworks, application cases, optimization method, and future directions, so as to provide students, researchers, and practitioners in related fields with a comprehensive reference. In detail, this book is organized as follows (as abstracted in Fig. 1): In Chap. 1, we have introduced the generation, development, trend, and industrial status of edge computing and given the brief of intelligent edge and edge intelligence. Next, we provide some fundamentals related to edge computing and AI in Chaps. 2 and 3, respectively. The following sections introduce the five enabling technologies, i.e., AI applications on edge (Chap. 4), AI inference in edge (Chap. 5), AI training at edge (Chap. 6), edge computing for AI (Chap. 7), and AI for optimizing edge (Chap. 8). Finally, we present lessons learned and discuss open challenges in Chap. 9 and conclude this book in Chap. 10. Therefore, the intended audience includes scientific researchers and industry professionals engaged in the field of edge computing and artificial

intelligence. Hopefully, this monograph can fill in the gaps in the architecture of edge AI and further expand the existing knowledge system in this field.

Tianjin, China — Xiaofei Wang
Tianjin, China — Yiwen Han
Shenzhen, China — Victor C. M. Leung
Singapore, Singapore — Dusit Niyato
Shenzhen, China — Xueqiang Yan
Guangzhou, Guangdong, China — Xu Chen

Acknowledgements

This work was supported by the National Key R&D Program of China (No. 2019YFB2101901 and No.2018YFC0809803), National Science Foundation of China (No. 61702364, No. 61972432, and No. U1711265), the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (No.2017ZT07X355), Chinese National Engineering Laboratory for Big Data System Computing Technology. It was also supported in part by Singapore NRF National Satellite of Excellence, Design Science and Technology for Secure Critical Infrastructure NSoE DeST-SCI2019-0007, A*STAR-NTU-SUTD Joint Research Grant Call on Artificial Intelligence for the Future of Manufacturing RGANS1906, WASP/NTU M4082187 (4080), Singapore MOE Tier 1 2017-T1-002-007 RG122/17, MOE Tier 2 MOE2014-T2-2-015 ARC4/15, Singapore NRF2015-NRF-ISF001-2277, and Singapore EMA Energy Resilience NRF2017EWT-EP003-041.

Contents

Part I Introduction and Fundamentals

1 Introduction 3
1.1 A Brief Introduction to Edge Computing 3
1.2 Trends in Edge Computing 6
1.3 Industrial Applications of Edge Computing 7
1.4 Intelligent Edge and Edge Intelligence 8
References 12

2 Fundamentals of Edge Computing 15
2.1 Paradigms of Edge Computing 15
2.1.1 Cloudlet and Micro Data Centers 16
2.1.2 Fog Computing 17
2.1.3 Mobile and Multi-Access Edge Computing (MEC) 17
2.1.4 Definition of Edge Computing Terminologies 18
2.1.5 Collaborative End–Edge–Cloud Computing 18
2.2 Hardware for Edge Computing 19
2.2.1 AI Hardware for Edge Computing 19
2.2.2 Integrated Commodities Potentially for Edge Nodes 21
2.3 Edge Computing Frameworks 22
2.4 Virtualizing the Edge 25
2.4.1 Virtualization Techniques 27
2.4.2 Network Virtualization 28
2.4.3 Network Slicing 28
2.5 Value Scenarios for Edge Computing 29
2.5.1 Smart Parks 29
2.5.2 Video Surveillance 30
2.5.3 Industrial Internet of Things 30
References 30

3 Fundamentals of Artificial Intelligence 33
3.1 Artificial Intelligence and Deep Learning 33

3.2 Neural Networks in Deep Learning 35
3.2.1 Fully Connected Neural Network (FCNN) 35
3.2.2 Auto-Encoder (AE) 36
3.2.3 Convolutional Neural Network (CNN) 36
3.2.4 Generative Adversarial Network (GAN) 37
3.2.5 Recurrent Neural Network (RNN) 38
3.2.6 Transfer Learning (TL) 39
3.3 Deep Reinforcement Learning (DRL) 41
3.3.1 Reinforcement Learning (RL) 41
3.3.2 Value-Based DRL 42
3.3.3 Policy-Gradient-Based DRL 42
3.4 Distributed DL Training 42
3.4.1 Data Parallelism 43
3.4.2 Model Parallelism 44
3.5 Potential DL Libraries for Edge 44
References 45

Part II Artificial Intelligence and Edge Computing

4 Artificial Intelligence Applications on Edge 51
4.1 Real-time Video Analytic 51
4.1.1 Machine Learning Solution 51
4.1.2 Deep Learning Solution 52
4.2 Autonomous Internet of Vehicles (IoVs) 54
4.2.1 Machine Learning Solution 54
4.2.2 Deep Learning Solution 55
4.3 Intelligent Manufacturing 56
4.3.1 Machine Learning Solution 57
4.3.2 Deep Learning Solution 58
4.4 Smart Home and City 59
4.4.1 Machine Learning Solution 60
4.4.2 Deep Learning Solution 61
References 62

5 Artificial Intelligence Inference in Edge 65
5.1 Optimization of AI Models in Edge 65
5.1.1 General Methods for Model Optimization 66
5.1.2 Model Optimization for Edge Devices 67
5.2 Segmentation of AI Models 69
5.3 Early Exit of Inference (EEoI) 71
5.4 Sharing of AI Computation 72
References 74

6 Artificial Intelligence Training at Edge 77
6.1 Distributed Training at Edge 77
6.2 Vanilla Federated Learning at Edge 82
6.3 Communication-Efficient FL 83

6.4 Resource-Optimized FL 85
6.5 Security-Enhanced FL 86
6.6 A Case Study for Training DRL at Edge 89
6.6.1 Multi-User Edge Computing Scenario 89
6.6.2 System Formulation 90
6.6.3 Offloading Strategy for Computing Tasks Based on DRL 92
6.6.4 Distributed Cooperative Training 93
References 93

7 Edge Computing for Artificial Intelligence 97
7.1 Edge Hardware for AI 97
7.1.1 Mobile CPUs and GPUs 97
7.1.2 FPGA-Based Solutions 99
7.1.3 TPU-Based Solutions 100
7.2 Edge Data Analysis for Edge AI 101
7.2.1 Challenge and Needs for Edge Data Process 101
7.2.2 Combination of Big Data and Edge Data Process 102
7.2.3 Architecture for Edge Data Process 103
7.3 Communication and Computation Modes for Edge AI 103
7.3.1 Integral Offloading 103
7.3.2 Partial Offloading 104
7.3.3 Vertical Collaboration 106
7.3.4 Horizontal Collaboration 108
7.4 Tailoring Edge Frameworks for AI 110
7.5 Performance Evaluation for Edge AI 112
References 113

8 Artificial Intelligence for Optimizing Edge 117
8.1 AI for Adaptive Edge Caching 117
8.1.1 Use Cases of DNNs 121
8.1.2 Use Cases of DRL 122
8.2 AI for Optimizing Edge Task Offloading 123
8.2.1 Use Cases of DNNs 124
8.2.2 Use Cases of DRL 125
8.3 AI for Edge Management and Maintenance 126
8.3.1 Edge Communication 126
8.3.2 Edge Security 127
8.3.3 Joint Edge Optimization 128
8.4 A Practical Case for Adaptive Edge Caching 129
8.4.1 Multi-BS Edge Caching Scenario 129
8.4.2 System Formulation 130
8.4.3 Weighted Distributed DQN Training and Cache Replacement 131
8.4.4 Conclusion for Edge Caching Case 132
References 132

Part III Challenges and Conclusions

9 Lessons Learned and Open Challenges 137
9.1 More Promising Applications 137
9.2 General AI Model for Inference 138
9.2.1 Ambiguous Performance Metrics 138
9.2.2 Generalization of EEoI 139
9.2.3 Hybrid Model Modification 139
9.2.4 Coordination Between AI Training and Inference 140
9.3 Complete Edge Architecture for AI 140
9.3.1 Edge for Data Processing 140
9.3.2 Microservice for Edge AI Services 142
9.3.3 Incentive and Trusty Offloading Mechanism for AI 142
9.3.4 Integration with "AI for Optimizing Edge" 143
9.4 Practical Training Principles at Edge 143
9.4.1 Data Parallelism Versus Model Parallelism 144
9.4.2 Training Data Resources 144
9.4.3 Asynchronous FL at Edge 145
9.4.4 Transfer Learning-Based Training 145
9.5 Deployment and Improvement of Intelligent Edge 146
References 147

10 Conclusions 149

Acronyms

A3C	Asynchronous advantage actor-critic
AC	Actor-critic
AE	Auto-encoder
AI	Artificial intelligence
A-LSH	Adaptive locality sensitive hashing
ALU	Arithmetic and logic unit
APU	AI processing unit
AR	Augmented reality
ASIC	Application specific integrated circuit
B/S	Browser/server
BNNS	Binarized neural networks
BS	Base station
C/S	Client/server
CDN	Content delivery network
CNN	Convolutional neural network
C-RAN	Cloud-radio access networks
CV	Computer vision
D2D	Device-to-device
DAD	Deep architecture decomposition
DAG	Directed acyclic graph
DBMS	Database management system
DDNNs	Distributed deep neural networks
DDoS	Distributed denial of service
DDPG	Deep deterministic policy gradient
DL	Deep learning
DNN	Deep neural network
Double-DQL	Double deep Q-learning
DP	Differential privacy
DQL	Deep Q-learning
DQN	Deep Q-learning network
DRAM	Dynamic RAM

DRL	Deep reinforcement learning
DSL	Domain-specific language
Dueling-DQL	Dueling deep Q-learning
DVFS	Dynamic voltage and frequency scaling
eBNNs	Embedded binarized neural networks
ECC	Edge Computing Consortium
ECSP	Edge computing service provider
EEoI	Early exit of inference
EH	Energy harvesting
ETSI	European Telecommunications Standards Institute
FAP	Fog radio access point
FCN	Fog computing node
FCNN	Fully connected neural network
FIFO	First in first out
FL	Federated learning
FTP	Fused Tile Partitioning
GAN	Generative adversarial network
GNN	Graph neural network
GPU	Graphics processing unit
IID	Independent and identically distributed
IOB	Input–output block
IoT	Internet of Things
IoVs	Internet of vehicles
KD	Knowledge distillation
kNN	k-nearest neighbor
LCA	Logic cell array
LFU	Least frequently used
LRU	Least recently used
LSTM	Long short-term memory
MAB	Multi-armed bandit
MDC	Micro data center
MDP	Markov decision process
MEC	Mobile (multi-access) edge computing
MLP	Multi-layer perceptron
NCS	Neural compute stick
NFV	Network functions virtualization
NLP	Natural language processing
NN	Neural network
NPU	Neural processing unit
P2P	Peer-to-peer
PC	Personal computer
PPO	Proximate policy optimization
QoE	Quality of experience
QoS	Quality of service
RAM	Random access memory

RAN	Radio access network
RL	Reinforcement learning
RNN	Recurrent neural network
RoI	Region-of-Interest
RPC	Remote procedure call
RRH	Remote radio head
RSU	Road-side unit
SDK	Software development kit
SDN	Software-defined network
SGD	Stochastic Gradient Descent
SINR	Signal-to-interference-plus-noise ratio
SNPE	Snapdragon neural processing engine
SNR	Signal-to-noise ratios
SRAM	Static random access memory
SVD	Singular value decomposition
TL	Transfer learning
TPU	Tensor processing unit
UE	User equipment
V2V	Vehicle-to-vehicle
VHDL	Very-high-speed integrated circuit hardware description language
VM	Virtual machine
VNF	Virtual network function
VPU	Vision processing unit
VR	Virtual reality
WLAN	Wireless local area network
ZB	Zettabytes

Part I
Introduction and Fundamentals

Chapter 1
Introduction

Abstract With the development of the Internet, there is a trend of blowout growth in network data. Meanwhile, the pursuit of the low latency of applications has also become a common user demand. Traditional cloud computing solves the problem of lack of resources faced by end devices through offloading data to the cloud, but it cannot meet the needs of people in the era of big data for computing efficiency. Therefore, edge computing came into being. By processing data in advance on devices close to the source of the data, edge computing reduces a lot of network transmission overhead, and also reduces response delay, while also having a positive effect on data privacy protection. The generation of edge computing is the result of the improvement of related technologies, and its development trend will also be the integration of other technologies. Among them, the combination of artificial intelligence technology and edge computing is an important development direction, and there is huge room for development in the future whether intelligent edge or edge intelligence.

1.1 A Brief Introduction to Edge Computing

With the proliferation of computing and storage devices, from server clusters in cloud data centers (the cloud) to personal computers and smartphones, further, to wearable and other Internet of Things (IoT) devices, we are now in an information-centric era in which computing is ubiquitous and computation services are overflowing from the cloud to the edge. According to a Cisco white paper [1], 50 billion IoT devices will be connected to the Internet by 2020. On the other hand, Cisco estimates that nearly 850 Zettabytes (ZB) of data will be generated each year outside the cloud by 2021, while global data center traffic is only 20.6 ZB [2]. This indicates that data sources for big data are also undergoing a transformation: from large-scale cloud data centers to an increasingly wide range of edge devices. However, existing cloud computing is gradually unable to manage these massively distributed computing power and analyze their data: (1) a large number of computation tasks need to be delivered to the cloud for processing [3], which undoubtedly poses serious challenges on network capacity and the computing

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_1

power of cloud computing infrastructures; (2) many new types of applications, e.g., cooperative autonomous driving, have strict or tight delay requirements that the cloud would have difficulty meeting since it may be far away from the users [4].

Therefore, edge computing [5, 6] emerges as an attractive alternative, especially to host computation tasks as close as possible to the data sources and end users. Certainly, edge computing and cloud computing are not mutually exclusive [7, 8]. Instead, the edge complements and extends the cloud. Compared with cloud computing only, the main advantages of edge computing combined with cloud computing are three folds: (1) **backbone network alleviation**, distributed edge computing nodes can handle a large number of computation tasks without exchanging the corresponding data with the cloud, thus alleviating the traffic load of the network; (2) **agile service response**, services hosted at the edge can significantly reduce the delay of data transmissions and improve the response speed; (3) **powerful cloud backup**, the cloud can provide powerful processing capabilities and massive storage when the edge cannot afford.

The leap from cloud computing to edge computing is another milestone in the evolution of the computing model. This can be found throughout the computing model. We know that the evolution of computing models and the development of computing equipment are actually synchronized. Therefore, based on the development history of computing equipment, we summarize the following evolution process for the computing model (Table 1.1):

Next, we make some brief introductions to these phases, respectively.

1. **Centralized processing model centered on the mainframe:** Due to the low computing power of the earliest computers and the huge and expensive computing equipment, almost all computing tasks depended on the central host computer, while other physical devices had no computing power and could only access the applications and data on the host computer to fulfill the different computing needs of each user.
2. **File-sharing computing model centered on PC/file server:** With the gradual development of the computing power of computers, the volume of computing equipment is reduced and the cost is reduced. PC gradually becomes the mainstream computing equipment. However, due to the backward storage medium, the data storage capacity of the PC is still insufficient. In this context, the original

Table 1.1 Evolution of the computing model

Time	Computing mode
1965–1985	Centralized processing model centered on the mainframe
1986–1990	File-sharing computing mode centered on PC/file server
1990–1996	Distributed computing model centered on C/S architecture
1996–	Distributed computing model with Web and B/S architecture
2000–	Centralized processing model centered on the mainframe
2005–	Distributed computing model with Grid, P2P, Cloud, and other technologies as cores
2015–	“End–edge–cloud” collaborative edge computing model

mainframe delegated most of the computing tasks, only to store a large amount of file data for each PC to access, thus forming a PC/file server as the core of the file-sharing computing model.

3. **Distributed computing model centered on C/S architecture:** With the emergence and popularity of database technology and the emergence of defects in the traditional file-sharing structure, the database server has replaced the traditional file server, which has resulted in the so-called C/S mode. In this mode, the server uses the DBMS to quickly respond to user requests and communicates through RPC or SQL.
4. **Distributed computing model with Web and B/S architecture:** With the rapid increase in the number of users, computing demand and data quantity, the traditional C/S mode keep developing and expanding in the development, forming two-layer C/S mode, three-layer C/S mode, and so on. At the same time, with the development and popularity of the Internet, browsers and web servers have been added to the C/S model and served as the client layer and middle layer in the C/S model, respectively, thus forming the so-called B/S architecture.
5. **Ubiquitous computing model with various mobile devices as the core:** After years of development, computers have gradually entered the laboratory into the office and even ordinary families, which has greatly promoted the development of the computer technology industry. However, the computer-centric computing model requires users to sit in front of the computer desktop to complete tasks, and cannot extend the business of computers to all aspects of people's lives, such as wearables, sensing devices for hazardous environments, etc. Therefore, the emergence of ubiquitous computing will enable information technology to be truly integrated into daily life.
6. **Distributed computing model with Grid, P2P, Cloud, and other technologies as cores:** The rapid development of information technology also makes users put forward more and higher requirements on the computing power of computers. To reduce the cost and make full use of idle computing resources while coping with the increasing data volume and high concurrent applications, a batch of distributed computing models such as grid computing, P2P, and cloud computing have become active in people's vision.
7. **"End–edge–cloud" collaborative edge computing model:** The advent of the Internet of Things technology has made computer technology truly integrated into human life. Nowadays, IoT has been upgraded from "things" and "things" to "people" and "things." This requires not only the computing model to have stronger computing and perception capabilities, but also a large amount of data generated by massive IoT devices that need to be addressed promptly. The traditional cloud computing model can no longer meet the needs of the "Internet of Everything" era, and the emergence of complementary edge computing will become the development direction of the next-generation network.

1.2 Trends in Edge Computing

The rise and development of edge computing are inseparable from the opportunity of a new round of technological change. These emerging technologies have helped edge computing develop from its architectural blueprint to industrial landing. At the same time, the maturity of edge computing technology and its standardization system will certainly provide the opportunities to achieve leapfrog development for these technologies. Therefore, the future development of edge computing is going to be integrated with the development of other technologies. We summarize the trend of edge computing into the following aspects: heterogeneous computing, edge intelligence, edge cloud collaboration, 5G + edge computing, and so on.

1. **Heterogeneous computing:** Heterogeneous computing can collaboratively use machines with different performance and structure to meet different computing needs and can obtain the maximum overall performance on heterogeneous platforms through algorithms. The approach can meet the demand of the heterogeneous computing platform and diverse data processing in the future. By introducing heterogeneous computing in edge computing, we can satisfy the needs of processing fragmented data and differentiated applications in edge services, and support flexible scheduling of computing resources based on improving computing resource utilization.
2. **Edge intelligence:** Edge intelligence, which uses edge computing to push artificial intelligence technology to the edge, is an application and expression of artificial intelligence. With the improvement of end hardware computing power, artificial intelligence has appeared in more and more terminal application scenarios. On the one hand, artificial intelligence deploying on the edge nodes can obtain richer data faster, which not only saves communication costs but also reduces response delays, and greatly expands the application scenarios of artificial intelligence; on the other hand, edge computing can use artificial intelligence technology to optimize edge-side resource scheduling decisions, help edge computing expand the business scope, and provide users with more efficient services. We have reason to believe that edge intelligence will become an important technology in future society.
3. **Edge cloud collaboration:** Edge computing is an extension of cloud computing, and it can complement each other with cloud computing. Cloud computing is good at global, non-real-time, long-cycle big data processing and analysis; edge computing does well in field-level, real-time, and short-cycle intelligent analysis. Therefore, in the face of similar applications related to artificial intelligence, we can deploy compute-intensive tasks in the cloud, and tasks that require a fast response are placed on the edge. At the same time, the edge can also preprocess the data sent to the cloud to further reduce network bandwidth consumption. Collaborative computing through edge and cloud can satisfy diverse needs and reduces computing costs and network bandwidth costs. Therefore, the coordinated development of edge computing and cloud computing not only has a huge boost to the development of these two technologies but also

provides a driving force for the development of other technologies, such as edge intelligence, IoT, etc.

4. **5G + edge computing:** 5G has three characteristics: ultra-high speed, large connection, and ultra-low latency and people achieve these characteristics relying on many advanced technologies including edge computing. 5G and edge computing are closely related, on the one hand, edge computing is an important component of the 5G network and effectively alleviate the problem of data explosion in the 5G era; on the other hand, 5G provides a good network foundation for the industrial deployment and development of the edge computing industry. Therefore, the development of 5G and edge computing is complementary and there is room for cooperation between the two in the support of the three major scenarios of 5G, and the development of network capabilities.

1.3 Industrial Applications of Edge Computing

Compared with cloud computing, edge computing is from cloud computing's "centralized resource sharing model" to "distributed mutual assistance sharing model" of each edge node. In the industry, service operators located at three different levels of "end–edge–cloud" have also proposed different solutions.

1. **Cloud service provider-based edge computing "cloud service drainage"**
   For example, the edge cloud nodes of Tencent, Baidu, and Ali, Google GKE On-Prem, Microsoft Azure Stack, and Huawei public cloud edge computing, aiming at the entrance of the cloud ecology, realize a small edge cloud on the edge cloud. To more complex content that will be available on the cloud.
2. **Site Facility Edge Alliance/Site Provider "Site + Computing Service"**
   Vapor, on behalf of more than a dozen companies, has jointly established the ecology, similar to the concept of China Tower Corporation providing many neutral sites, and connecting edge sites into a network through various mechanisms such as machine rooms and containers.
3. **Fixed operator-oriented edge computing "fixed connection + computing services"**
   China's Edge Computing Alliance (ECC) is a fixed network edge computing initiated by Huawei's fixed network and interacting extensively with the Industrial Internet. The main scenarios are enterprise switching equipment and fixed access edge equipment.
4. **Mobile operator-centric edge computing (MEC) "mobile connectivity + computing services"**
   Edge computing was called "mobile edge computing" in 2014. The early scenario was to do cache locally at the mobile base station (but failed to promote commercial use at that time because the content distribution and management base station did not perceive it), which later evolved into a packet mobile gateway PGW, local distribution, content caching and acceleration, docking to

fixed networks, IPTV, multicast, and other forms are commercially available (e.g., Huawei is commercially available in multiple regions around the world). In 2017, Huawei's wireless core network and other parties proposed that ETSI change the full name of the MEC to "multi-access edge computing," and in 2018, combined with the 5G network architecture, it was specified in the 3GPP international standard SA2 (Network Architecture Group) that MEC is deployed in UPF position and various enhancements to UPF. With 5G, MEC is also moving towards fixed-mobile convergence and multiple access.

5. **Edge computing in the form of a network or self-organizing network formed by industry/enterprise/traffic road administration**
   "Industry self-built connection + computing services" can be expanded to decompose, and some of them are equivalent to the fourth type of mobile operator-based or third type fixed-operator networks (the spectrum can be changed to unlicensed) based on the technology protocol stack. Some of them, such as LoRa or Wi-Fi, are used for free networking. With the implementation of cloud service providers, site service providers, the Internet of Things, and terminal forms, the protocol stack is similar to the first/second/sixth category.
6. **Edge computing for Terminal/CPE, IoT GW, and vehicle**
   The "near-end computing service" is to perform calculation processing locally in time, and then connect to the macro network to assist the upper-level node or surrounding nodes.

1.4 Intelligent Edge and Edge Intelligence

As a typical and more widely used new form of applications [9], various deep learning-based intelligent services and applications have changed many aspects of people's lives due to the great advantages of Deep Learning (DL) in the fields of Computer Vision (CV) and Natural Language Processing (NLP) [10]. These achievements are not only derived from the evolution of AI but also inextricably linked to increasing data and computing power. Nevertheless, for a wider range of application scenarios, such as smart cities, Internet of Vehicles (IoVs), etc., there are only a limited number of intelligent services offered due to the following factors.

- **Cost:** training and inference of AI models in the cloud requires devices or users to transmit massive amounts of data to the cloud, thus consuming a large amount of network bandwidth;
- **Latency:** the delay to access cloud services is generally not guaranteed and might not be short enough to satisfy the requirements of many time-critical applications such as cooperative autonomous driving [11];
- **Reliability:** most cloud computing applications relies on wireless communications and backbone networks for connecting users to services, but for many industrial scenarios, intelligent services must be highly reliable, even when network connections are lost;

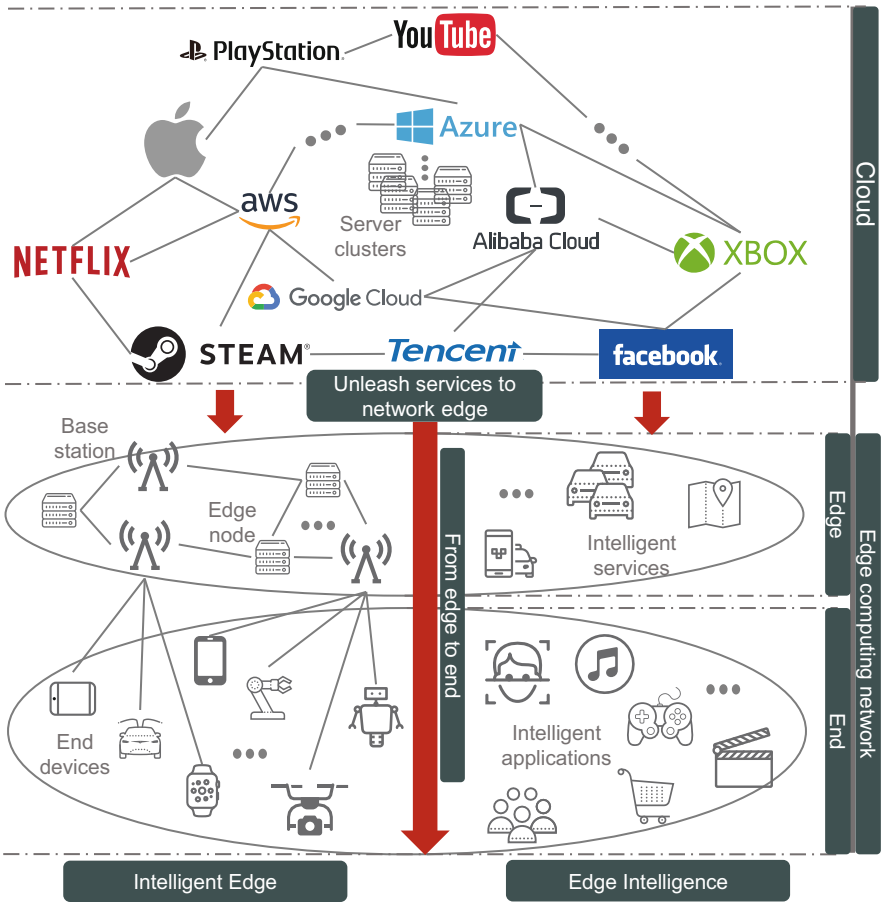

Fig. 1.1 Edge intelligence and intelligent edge

- **Privacy:** the data required for AI might carry a lot of private information, and privacy issues are critical to areas such as smart home and cities.

Since the edge is closer to users than the cloud, edge computing is expected to solve many of these issues. In fact, edge computing is gradually being combined with Artificial Intelligence (AI), benefiting each other in terms of the realization of *edge intelligence* and *intelligent edge* as depicted in Fig. 1.1. Edge intelligence and intelligent edge are not independent of each other. Edge intelligence is the goal, and the AI services in intelligent edge are also a part of edge intelligence. In turn, intelligent edge can provide higher service throughput and resource utilization for edge intelligence.

To be specific, on one hand, edge intelligence is expected to push AI computations from the cloud to the edge as much as possible, thus enabling various

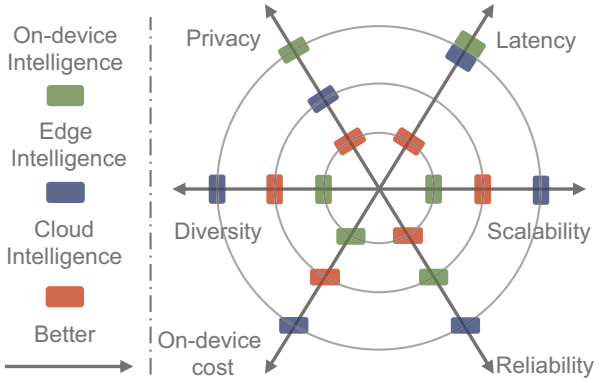

Fig. 1.2 Capabilities comparison of cloud, on-device, and edge intelligence

distributed, low-latency, and reliable intelligent services. As shown in Fig. 1.2, the advantages include: (1) AI services are deployed close to the requesting users, and the cloud only participates when additional processing is required [12], hence significantly reducing the latency and cost of sending data to the cloud for processing; (2) since the raw data required for AI services is stored locally on the edge or user devices themselves instead of the cloud, protection of user privacy is enhanced; (3) the hierarchical computing architecture provides more reliable AI computation; (4) with richer data and application scenarios, edge computing can promote the pervasive application of AI and realize the prospect of "providing AI for every person and every organization at everywhere" [13]; (5) diversified and valuable AI services can broaden the commercial value of edge computing and accelerate its deployment and growth.

On the other hand, intelligent edge aims to incorporate AI into the edge for dynamic, adaptive edge maintenance and management. With the development of communication technology, network access methods are becoming more diverse. At the same time, the edge computing infrastructure acts as an intermediate medium, making the connection between ubiquitous end devices and the cloud more reliable and persistent [14]. Thus the end devices, edge, and cloud are gradually merging into a community of shared resources. However, the maintenance and management of such a large and complex overall architecture (community) involving wireless communication, networking, computing, storage, etc., is a major challenge [15]. Typical network optimization methodologies rely on fixed mathematical models; however, it is difficult to accurately model rapidly changing edge network environments and systems. DL which is an important technology of AI recently is expected to deal with this problem: when faced with complex and cumbersome network information, DL can rely on its powerful learning and reasoning ability to extract valuable information from data and make adaptive decisions, achieving intelligent maintenance and management accordingly.

Therefore, considering that edge intelligence and intelligent edge, i.e., Edge AI, together face some of the same challenges and practical issues in multiple aspects, we identify the following five technologies that are essential for Edge AI:

1. *AI applications on Edge*: Technical frameworks for systematically organizing edge computing and AI to provide intelligent services;
2. *AI inference in Edge*: Focusing on the practical deployment and inference of AI in the edge computing architecture to fulfill different requirements, such as accuracy and latency;
3. *Edge computing for AI*: Which adapts the edge computing platform in terms of network architecture, hardware, and software to support AI computation;
4. *AI training at Edge*, training AI models for edge intelligence at distributed edge devices under resource and privacy constraints;
5. *AI for optimizing Edge*: Application of AI for maintaining and managing different functions of edge computing networks (systems), e.g., edge caching [16], computation offloading[17].

As illustrated in Fig. 1.3, "DL applications on Edge" and "DL for optimizing edge" correspond to the theoretical goals of edge intelligence and intelligent edge, respectively. To support them, various DL models should be trained by intensive computation at first. In this case, for the related works leveraging edge computing resources to train various DL models, we classify them as "DL training at Edge." Second, to enable and speed up Edge AI services, we focus on a variety of techniques supporting the efficient inference of DL models in edge computing frameworks and networks, called "DL inference in Edge." At last, we classify all techniques, which adapts edge computing frameworks and networks to better serve Edge AI, as "Edge computing for DL."

To the best of our knowledge, existing articles that are most related to our work include [18–21]. Different from our more extensive coverage of Edge AI, [18] is focussed on the use of machine learning (rather than DL) in edge intelligence for wireless communication perspective, i.e., training machine learning at the network

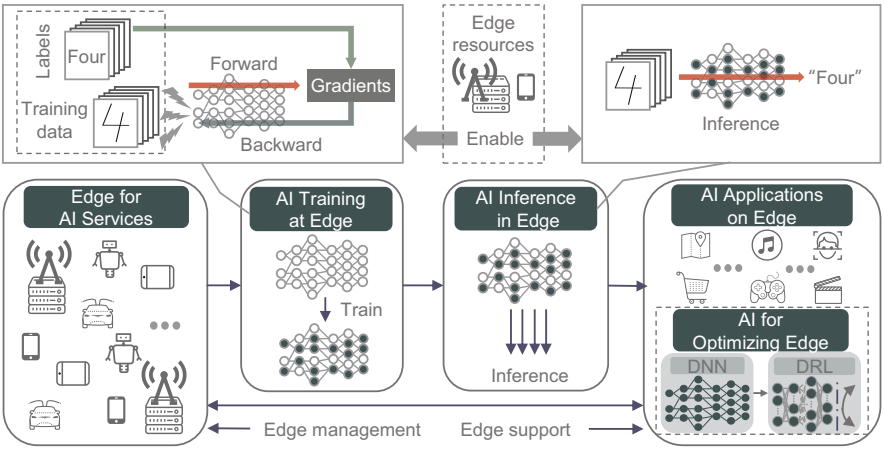

Fig. 1.3 Landscape of Edge AI according to the proposed taxonomy

edge to improve wireless communication. Besides, discussions about DL inference and training are the main contribution of [19–21]. Different from these works, this survey focuses on these respects: (1) comprehensively consider deployment issues of AI which is mainly DL, by edge computing, spanning networking, communication, and computation; (2) investigate the holistic technical spectrum about the convergence of DL and edge computing in terms of the five enablers; (3) point out that DL and edge computing are beneficial to each other and considering only deploying DL on the edge is incomplete.

In summary, with the development of the network, explosive growth in Internet data has provided a good development environment for edge computing. At the same time, the emergence of edge computing has also promoted a new revolution in computing models and network structures. In our opinion, edge computing is more like a method or a tool for solving problems, which can be combined with any existing technology and improve its performance. Therefore, whether it is in industrial production, entertainment, or deep learning, edge computing can play an important role and bring us more outstanding results.

References

1. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf
2. Cisco Global Cloud Index: Forecast and Methodology. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html
3. M.V. Barbera, S. Kosta, A. Mei et al., To offload or not to offload? The bandwidth and energy costs of mobile cloud computing, in *2013 IEEE Conference on Computer Communications (INFOCOM 2013)* (2013), pp. 1285–1293
4. W. Hu, Y. Gao, K. Ha et al., Quantifying the impact of edge computing on mobile applications, in *Proceedings of the Seventh ACM SIGOPS Asia-Pacific Workshop System (APSys 2016)* (2016), pp. 1–8
5. Mobile-Edge Computing—Introductory Technical White Paper, ETSI. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing_-_Introductory_Technical_White_Paper_V1%2018-09-14.pdf
6. W. Shi, J. Cao et al., Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
7. B.A. Mudassar, J.H. Ko, S. Mukhopadhyay, Edge-cloud collaborative processing for intelligent internet of things, in *Proceedings of the 55th Annual Design Automation Conference (DAC 2018)* (2018), pp. 1–6
8. A. Yousefpour, C. Fung, T. Nguyen et al., All one needs to know about fog computing and related edge computing paradigms: a complete survey. J. Syst. Architect. **98**, 289–330 (2019)
9. J. Redmon, S. Divvala et al., You only look once: unified, real-time object detection, in *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)* (2016), pp. 779–788
10. J. Schmidhuber, Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
11. H. Khelifi, S. Luo, B. Nour et al., Bringing deep learning at the edge of information-centric internet of things. IEEE Commun. Lett. **23**(1), 52–55 (2019)

12. Y. Kang, J. Hauswald, C. Gao et al., Neurosurgeon: collaborative intelligence between the cloud and mobile edge, in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017)* (2017), pp. 615–629
13. Democratizing AI. https://news.microsoft.com/features/democratizing-ai/
14. Y. Yang, Multi-tier computing networks for intelligent IoT. Nat. Electron. **2**(1), 4–5 (2019)
15. C. Li, Y. Xue, J. Wang et al., Edge-oriented computing paradigms: a survey on architecture design and system management. ACM Comput. Surv. **51**(2), 1–34 (2018)
16. S. Wang, X. Zhang, Y. Zhang et al., A survey on mobile edge networks: convergence of computing, caching and communications. IEEE Access (5), 6757–6779 (2017)
17. T.X. Tran, A. Hajisami et al., Collaborative mobile edge computing in 5G networks: new paradigms, scenarios, and challenges. IEEE Commun. Mag. **55**(4), 54–61 (2017)
18. J. Park, S. Samarakoon, M. Bennis, M. Debbah, Wireless network intelligence at the edge. Proc. IEEE **107**(11), 2204–2239 (2019)
19. Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, J. Zhang, Edge intelligence: paving the last mile of artificial intelligence with edge computing. Proc. IEEE **107**(8), 1738–1762 (2019)
20. J. Chen, X. Ran, Deep learning with edge computing: a review. Proc. IEEE **107**(8), 1655–1674 (2019)
21. W.Y.B. Lim, N.C. Luong, D.T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato et al., Federated learning in mobile edge networks: a comprehensive survey (2019). arXiv:1909.11875

Chapter 2
Fundamentals of Edge Computing

Abstract Edge computing has become an important solution to break the bottleneck of emerging technologies by virtue of its advantages of reducing data transmission, improving service latency, and easing cloud computing pressure. At the same time, the emergence of edge computing has spawned a large number of new technologies and promoted the updating and application of some existing technologies, such as the improvement of end device hardware capabilities and the expansion of application scenarios of virtualization technologies. The edge computing architecture will become an important complement to the cloud, even replacing the role of the cloud in some scenarios. However, the status of cloud computing will not be completely replaced by edge computing, because cloud computing can process some computation-intensive tasks that edge devices cannot deal with, relying on its rich computing power and storage resources. Therefore, the combination of cloud computing and edge computing can satisfy the requirements of more diverse application scenarios and bring a more convenient experience to users.

2.1 Paradigms of Edge Computing

In the development of edge computing, there have been various new technologies aimed at working at the edge of the network, with the same principles but different focuses, such as Cloudlet [1], Micro Data Centers (MDCs) [2], Fog Computing [3, 4], and Mobile Edge Computing [5] (viz., Multi-access Edge Computing [6] now). However, the edge computing community has not yet reached a consensus on the standardized definitions, architectures and protocols of edge computing [7]. We use a common term "edge computing" for this set of emerging technologies. In this section, different edge computing concepts are introduced and differentiated. Table 2.1 compares some features of fog computing, mobile edge computing (MEC), and Cloudlet.

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_2

Table 2.1 Comparison of Cloudlet, fog computing, and MEC

Characteristic	Cloudlet	Fog computing	MEC
Context aware	Low	Mid	High
Inter-node communication	Partial support	Support	Partial support
Accessibility	Single hop	Single or multiple hops	Single hop
Access mechanism	Wi-Fi	Bluetooth, Wi-Fi, mobile network	Mobile network
Node location	Local/outdoor installation	Between the end device and the cloud	Radio network controller/macro basestation
Node device instance	Data center in a box	Routers, set-top boxes, switches, IoT gateways	Edge server running in base station

2.1.1 Cloudlet and Micro Data Centers

Cloudlet was first proposed by a team from CMU in 2009 as an early example form of edge computing. Cloudlet is a network architecture element that combines mobile computing and cloud computing. It represents the middle layer of the three-tier architecture, i.e., mobile devices, the micro cloud, and the cloud. Its highlights are efforts to (1) define the system and create algorithms that support low-latency edge–cloud computing, and (2) implement related functionality in open source code as an extension of Open Stack cloud management software [1]. Cloudlet is regarded as a "data center in a box" in an attempt to make cloud computing closer to the users. It can provide real-time resources to end devices through a WLAN network by running virtual machines on devices. Cloudlet is composed of a set of resource-rich, multi-core computers that have high-speed internet connectivity and high-bandwidth wireless LANs for nearby end devices. For security purposes, cloudlets are packaged in a tamper-resistant box to ensure the security of unmonitored areas.

The concept of MDCs was first proposed by Microsoft Research as an extension of hyper-scale cloud data centers. Similar to Cloudlets, MDCs [2] are also designed to complement the cloud. The idea is to package all the computing, storage, and networking equipment needed to run customer applications in one enclosure, as a stand-alone secure computing environment, for applications that require lower latency or end devices with limited battery life or computing abilities. Meanwhile, MicroData Centers have a certain degree of flexibility and scalability in terms of capacity and latency and can change its size according to network bandwidth and user needs.

2.1.2 Fog Computing

Fog computing is a concept proposed by Cisco[3] in 2012 as an extension of cloud computing from the network core to the network edge. The OpenFog Consortium[8] which is the main promoter of fog computing defines fog computing as "a horizontal system-level architecture that distributes computing, storage, control, and networking functions closer to the users along a cloud-to-thing continuum." One of the highlights of fog computing is that it assumes a fully distributed multi-tier cloud computing architecture with billions of devices and large-scale cloud data centers [3, 4]. The intermediary computing units in fog computing between the end device and the cloud data center are called fog computing nodes (FCN). FCN is heterogeneous in nature, including but not limited to routers, set-top boxes, switches, IoT gateways, etc. This heterogeneous feature of FCN supports devices at different protocol layers and even supports non-IP access technologies to communicate between FCN and end devices. At the same time, the fog abstraction layer is used to hide the heterogeneity of the nodes, thereby providing functions such as data management and communication services between the end device and the cloud. However, fog computing cannot run on its own without cloud computing and exists only as an extension of cloud computing. While cloud and fog paradigms share a similar set of services, such as computing, storage, and networking, the deployment of fog is targeted to specific geographic areas. In addition, fog is designed for applications that require real time responding with less latency, such as interactive and IoT applications. Unlike Cloudlet, MDCs and MEC, fog computing is more focused on IoTs.

2.1.3 Mobile and Multi-Access Edge Computing (MEC)

MEC originally appeared as the concept of Mobile Edge Computing and be standardized by the Mobile Edge Computing Specification Working Group of the European Telecommunications Standards Institute (ETSI) in 2014. However, with the actual needs constantly changing, ETSI extends the concept of MEC to Multi-access Edge Computing in 2016, which means further extending edge computing from telecommunications cellular networks to other wireless access networks. Mobile Edge Computing places computing capabilities and service environments at the edge of cellular networks [5]. It is designed to provide lower latency, context and location awareness, and higher bandwidth. Deploying edge servers on cellular Base Stations (BSs) allows users to deploy new applications and services flexibly and quickly. The European Telecommunications Standards Institute (ETSI) further extends the terminology of MEC from Mobile Edge Computing to Multi-access Edge Computing by accommodating more wireless communication technologies, such as Wi-Fi [6]. Multi-access edge computing provides application developers and content providers with IT service environments and computing capabilities at

the edge of the network, thereby realizing the characteristics of ultra-low latency, high bandwidth, and real-time access for related applications. The expansion of this concept includes the requirements of non-mobile networks, which is more in line with the requirements of today's application scenarios and the development trend of edge computing.

2.1.4 Definition of Edge Computing Terminologies

The definition and division of edge devices are ambiguous in most literature (the boundary between edge nodes and end devices is not clear). For this reason, as depicted in Fig. 1.1, we further divide common edge devices into end devices and edge nodes: the "end devices" (end level) is used to refer to mobile edge devices (including smartphones, smart vehicles, etc.) and various IoT devices, and the "edge nodes" (edge level) include Cloudlets, Road-Side Units (RSUs), Fog nodes, edge servers, MEC servers and so on, namely servers deployed at the edge of the network.

2.1.5 Collaborative End–Edge–Cloud Computing

While cloud computing is created for processing computation-intensive tasks, such as DL, it cannot guarantee the delay requirements throughout the whole process from data generation to transmission to execution. Moreover, independent processing on the end or edge devices is limited by their computing capability, power consumption, and cost bottleneck. Therefore, collaborative end–edge–cloud computing for DL [9], abstracted in Fig. 2.1, is emerging as an important trend as depicted in Fig. 2.2. In this novel computing paradigm, computation tasks with lower computational intensities, generated by end devices, can be executed directly at the end devices or offloaded to the edge, thus avoiding the delay caused by sending data to the cloud. For a computation-intensive task, it will be reasonably segmented and dispatched separately to the end, edge and cloud for execution, reducing the execution delay of the task while ensuring the accuracy of the results [9–11]. The focus of this collaborative paradigm is not only the successful completion of tasks

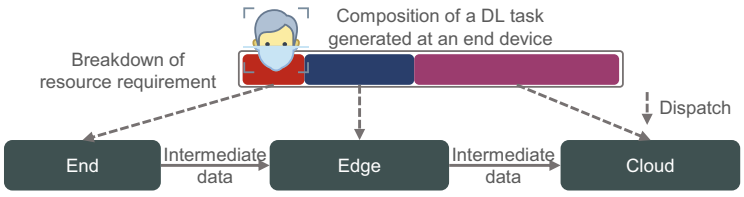

Fig. 2.1 A sketch of collaborative end–edge–cloud DL computing

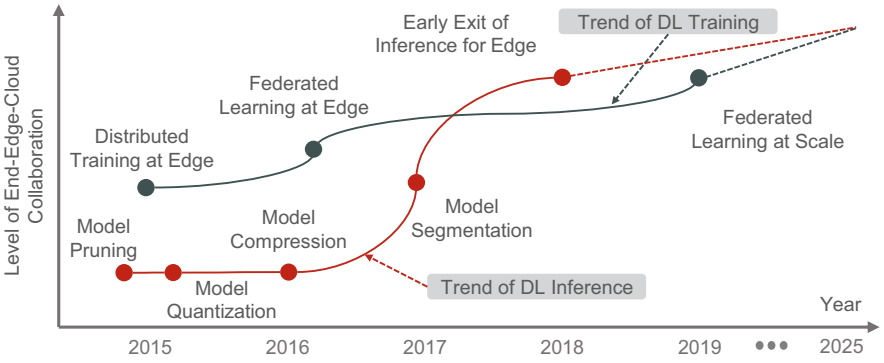

Fig. 2.2 Computation collaboration is becoming more important for DL with respect to both training and inference

but also achieving the optimal balance of equipment energy consumption, server loads, transmission and execution delays.

2.2 Hardware for Edge Computing

The generation of edge computing is inseparable from the improvement of hardware capabilities in recent years, and it is no doubt that further improvement of hardware capabilities is going to promote the development and application of edge intelligence. The development of edge intelligence is both a software level improvement and a hardware level improvement according to the history of artificial intelligence and edge computing. Therefore, we should understand the current development status of edge device hardware if we want to grasp the future development direction of edge intelligence. In this section, we discuss potential enabling hardware of edge intelligence, i.e., customized AI chips and commodities for both end devices and edge nodes. Besides, edge–cloud systems for DL are introduced as well (listed in Table 2.2).

2.2.1 AI Hardware for Edge Computing

Emerged edge AI hardware can be classified into three categories according to their technical architecture: (1) Graphics Processing Unit (GPU) -based hardware. Unlike CPU which has strong general-purpose characteristics, GPU was originally designed to process a large number of concurrent computing tasks. Therefore, GPU has many arithmetic and logic units (ALU) and very few caches whose purpose is not to save the data that needs to be accessed later but improve services

Table 2.2 Summary of edge computing AI hardware and systems

	Owner	Production	Feature
Integrated commodities	Microsoft	Data Box Edge [12]	Competitive in data preprocessing and data transmission
	Intel	Movidius Neural Compute Stick	Prototype on any platform with plug-and-play simplicity
	NVIDIA	Jetson	Easy-to-use platforms that runs in as little as 5 W
	Huawei	Atlas Series [13]	An all-scenario AI infrastructure solution that bridges "device, edge, and cloud"
AI hardware for edge computing	Qualcomm	Snapdragon 8 Series [14]	Powerful adaptability to major DL frameworks
	HiSilicon	Kirin 600/900 Series [15]	Independent NPU for DL computation
	HiSilicon	Ascend Series [16]	Full coverage—from the ultimate low energy consumption scenario to high computing power scenario
	MediaTek	Helio P60 [17]	Simultaneous use of GPU and NPU to accelerate neural network computing
	NVIDIA	Turing GPUs [18]	Powerful capabilities and compatibility but with high energy consumption
	Google	TPU [19]	Stable in terms of performance and power consumption
	Intel	Xeon D-2100	Optimized for power- and space-constrained cloud–edge solutions
	Samsung	Exynos 9820	Mobile NPU for accelerating AI tasks
Edge computing frameworks	Huawei	KubeEdge [20]	Native support for edge–cloud collaboration
	Baidu	OpenEdge [21]	Computing framework shielding and application production simplification
	Microsoft	Azure IoT Edge [22]	Remotely edge management with zero-touch device provisioning
	Linux Foundation	EdgeX [23]	IoT edge across the industrial and enterprise use cases
	Linux Foundation	Akraino Edge Stack [24]	Integrated distributed cloud–edge platform
	NVIDIA	NVIDIA EGX [26]	Real-time perception, understanding, and processing at the edge
	Amazon	AWS IoT Greengrass	Tolerance to edge devices even with intermittent connectivity
	Google	Google Cloud IoT	Compatible with Google AI products, such as TensorFlow Lite and Edge TPU

for threads. Graphics Processing Unit (GPU)-based hardware architecture is the main architecture in the field of artificial intelligence hardware today because of its outstanding computing power, and it tends to have good compatibility and performance, but generally consumes more energy, e.g., NVIDIA' GPUs based on Turing architecture[18]. (2) Field Programmable Gate Array (FPGA)—based hardware [25, 26]. People use the concept of logic cell array (LCA) in FPGA which includes three parts: Configurable Logic Block (CLB), Input–Output Block (IOB), and Interconnect. FPGA is a programmable device that can be used to design ASIC circuits and has the characteristics of low development costs, short design cycles, low energy consumption, reusable, and easy to learn and use. FPGA-based hardware architecture also has a good development trend in the direction of edge intelligence due to the characteristics of energy saving and require less computation resources, although it has the shortcoming of worse compatibility and limited programming. (3) Application-Specific Integrated Circuit (ASIC) -based hardware. The most obvious feature of ASIC is that it is developed for specific needs, so it has advantages incomparable to other hardware in specific scenarios. Therefore, during the development of edge intelligence, ASIC-based hardware architecture can help edge intelligence achieve better results which could be more stable in terms of performance and power consumption due to the custom design in specific scenarios, such as Google's TPU [19] and HiSilicon's Ascend series [16].

As smartphones represent the most widely deployed edge devices, chips for smartphones have undergone rapid developments, and their capabilities have been extended to the acceleration of AI computing. To name a few, Qualcomm first applies AI hardware acceleration [14] in Snapdragon and releases Snapdragon Neural Processing Engine (SNPE) SDK, which supports almost all major DL frameworks. Compared to Qualcomm, HiSilicon's 600 series and 900 series chips [15] do not depend on GPUs. Instead, they incorporate an additional Neural Processing Unit (NPU) to achieve fast calculation of vectors and matrices, which greatly improves the efficiency of DL. Compared to HiSilicon and Qualcomm, MediaTek's Helio P60 not only uses GPUs but also introduces an AI Processing Unit (APU) to further accelerate neural network computing [17]. Performance comparison of most commodity chips with respect to DL can be found in [27], and more customized chips of edge devices will be discussed in detail later.

2.2.2 Integrated Commodities Potentially for Edge Nodes

Edge nodes are expected to have computing and caching capabilities and to provide high-quality network connection and computing services near end devices. Compared to most end devices, edge nodes have more powerful computing capability to process tasks. On the other side, edge nodes can respond to end devices more quickly than the cloud. Therefore, by deploying edge nodes to perform the computation task, the task processing can be accelerated while ensuring accuracy. In addition, edge nodes also have the ability to cache, which can improve the response

time by caching popular contents. Most edge nodes have the ability to process preliminary Deep Learning (DL) inference and then transfer to the cloud for further processing.

In order to learn more about edge devices, we next introduce some related examples, including Data Box Edge [12], Jetson, and Atlas [13]. Data Box Edge is a local device released by Microsoft with AI-enabled edge computing capabilities. It can act as a storage gateway and create a link between user sites and Azure storage, which makes it easy to move data in and out of Azure storage using local network sharing. At the same time, Data Box Edge provides a technology platform through IoT Edge, enabling users to deploy Azure services and applications to the edge. The Jetson series is an embedded system designed by NVIDIA for a new generation of autonomous machines. It is an AI platform and mainly used in autonomous machines, high-definition sensors, and video analysis. There are currently four product types in the Jetson series, including JetsonNano, Jetson TX2, Xavier NX, and AGX Xavier. These products have different positioning, which can meet the needs of different levels in the industry from face recognition to autonomous driving. The Atlas series is a new generation of intelligent cloud hardware platform launched by Huawei, which has the main features of resource pooling, heterogeneous computing, and second-level deployment. The Atlas series includes an acceleration module for the end side, an acceleration card for the data center side, a smart station for the edge side, and a one-stop AI platform positioned in the enterprise field. Therefore, the Atlas series forms a complete AI solution while enriching the product form.

2.3 Edge Computing Frameworks

Solutions for edge computing systems are blooming. For DL services with complex configuration and intensive resource requirements, edge computing systems with advanced and excellent microservice architecture are the future development direction. In the environment of edge computing, end devices will generate a large amount of heterogeneous data that will be processed by various devices for different applications with diversified requirements. Therefore, it is a difficult problem on how to design an edge computing framework to ensure computing tasks feasibility, application reliability, and maximize resource utilization. At the same time, it should be different for the edge computing frameworks in different application scenarios, because there are different demands in diverse scenarios. In order to have a deeper understanding of the development of edge computing, we summarize some existing edge computing frameworks in Table 2.3 which mainly compares some edge computing platforms based on the following indicators:

- **Design goals:** The goals of the edge computing framework reflect the problem areas they are aimed at and influence the framework's system structure and functional design.

Table 2.3 Summary of edge computing Frameworks

	Custodian or owner	Design goals	Target users	Scalability	System features	Application scenarios
EdgeX Foundry	Linux Foundation	Interoperability of IoT devices	No limitation	Support	Provide users with a common API to control the connected devices, which is convenient for large-scale monitoring and control of IoT devices	Industrial internet of things, intelligent transportation, smart factory
Apache Edgent	Apache Software Foundation	Efficient analysis and processing of data from edge devices	No limitation	\	Provide data processing APIs to meet the actual needs of data processing in networking applications	IoT, machine health, analysis logs, text and other types of data
CORD	Open Network Foundation (ONF)	Restructure existing network edge infrastructure and build data centers that can flexibly provide computing and network services	Network operators	Support	No need for users to provide computing resources and build computing platforms and independent of geographical location	Mobile device applications, VR, AR
Akraino edge stack	Linux Foundation	Optimize network construction and management of edge infrastructure	Network operators	\	Provide edge–cloud services based on use cases	Edge video processing, smart cities, smart transportation
StarlingX	OpenStack Founda-tion/WindRiver, Intel	Optimization for low-latency and high-performance applications	Network operators, enterprise	Support	Compatibility with different open source components	Services with high latency requirements such as the industrial internet of things, telecommunications, and video services

(continued)

Table 2.3 (continued)

	Custodian or owner	Design goals	Target users	Scalability	System features	Application scenarios
KubeEdge	CNCF Foundation/Huawei	Extend containerized application orchestration capabilities to Edge hosts	No limitation	Support	Fully open, extensible, portable framework based on Kubernetes	Smart park, autonomous driving, industrial internet, interactive live broadcast
k3s	Rancher	Run small Kubernetes clusters on edge nodes of multiple architectures	No limitation	\	Lightweight and low resource requirements	Smart cars, ATMs, smart meters
Baetyl	Linux Foundation/Baidu	Extend cloud computing capabilities to user sites	No limitation	Support	Shield computing framework, simplify application production, deploy services on demand, and support multiple platforms	Autonomous driving, smart factory
AWS IoT Greengrass	AWS	Extend cloud capabilities to local devices	No limitation	\	Provide AWS cloud services and related technical support	Internet of vehicles, industrial internet of things
Azure IoT edge	Microsoft Azure	Extend cloud capabilities to edge devices	Enterprise	\	Supported by strong Azure cloud services, especially artificial intelligence and data analysis services	Smart factory, smart irrigation system
Link IoT edge	Aliyun	Extend cloud capabilities to the edge	IoT Developers	Support	Provide Aliyun cloud services and related technical support	Future hotels, edge gateways

- **Target users:** The target users of the edge computing framework are the demand side at the beginning of the platform design, and the design goals are established according to the needs of the target users. Some frameworks are provided to network operators, and some platforms have no restriction, that is any user can deploy the framework on the edge device.
- **Scalability:** Some edge computing frameworks need to support certain scalability in order to meet the needs of users in the future. Therefore, it is very attractive for many users to choose an edge computing framework with good scalability.
- **System features:** The system features of the edge computing framework are a reflection of the design goals, so users can deploy the framework in specific scenarios according to the system features.
- **Application scenarios:** Application scenarios are where edge computing platforms generate value and good application scenarios can greatly tap the platform's potential.

Currently, Kubernetes is as a mainstream container-centric system for the deployment, maintenance, and scaling of applications in cloud computing [28]. Based on Kubernetes, Huawei develops its edge computing solution "KubeEdge" [20] for networking, application deployment, and metadata synchronization between the cloud and the edge (also supported in Akraino Edge Stack [24]). "OpenEdge" [21] focus on shielding computing framework and simplifying application production. For IoT, Azure IoT Edge [22] and EdgeX [23] are devised for delivering cloud intelligence to the edge by deploying and running AI on cross-platform IoT devices (Table 2.4).

2.4 Virtualizing the Edge

Virtualization technology is one of the important promoters of the rapid development of cloud computing, but can virtualization technology still make a difference when edge computing and AI become the hot direction of future development?

The requirements of virtualization technology for integrating edge computing and AI reflect in the following aspects: (1) The resource of edge computing is limited. Edge computing cannot provide that resources for AI services as the cloud does. Virtualization technologies should maximize resource utilization under the constraints of limited resources; (2) AI services rely heavily on complex software libraries. The versions and dependencies of these software libraries should be taken into account carefully. Therefore, virtualization catering to Edge AI services should be able to isolate different services. Specifically, the upgrade, shutdown, crash, and high resource consumption of a single service should not affect other services; (3) The service response speed is critical for Edge AI. Edge AI requires not only the computing power of edge devices but also the agile service response that the edge computing architecture can provide.

Table 2.4 Potential DL frameworks for edge computing

Framework	Owner	Support edge	Android	iOS	Arm-Linux	FPGA	DSP	GPU	Mobile GPU	Support training
CNTK [29]	Microsoft	×	×	×	×	×	×	✓	×	✓
Chainer [30]	Preferred networks	×	×	×	×	×	×	✓	×	✓
TensorFlow [31]	Google	✓	×	×	✓	×	×	✓	×	✓
DL4J [32]	Skymind	✓	✓	×	✓	×	×	✓	×	✓
TensorFlow Lite [33]	Google	✓	✓	×	✓	×	×	✓	✓	×
MXNet [34]	Apache incubator	✓	✓	✓	✓	×	×	✓	×	✓
(Py)Torch [35]	Facebook	✓	✓	✓	✓	✓	×	✓	×	✓
CoreML [36]	Apple	✓	×	✓	×	×	×	×	✓	×
SNPE	Qualcomm	✓	✓	×	✓	×	✓	×	✓	×
NCNN [37]	Tencent	✓	✓	✓	✓	×	×	×	✓	×
MNN [38]	Alibaba	✓	✓	✓	✓	×	×	×	✓	×
Paddle-mobile [39]	Baidu	✓	✓	✓	✓	✓	×	×	✓	×
MACE [40]	XiaoMi	✓	✓	✓	✓	×	×	×	✓	×
FANN [41]	ETH Zrich	✓	×	×	✓	×	×	×	×	✓

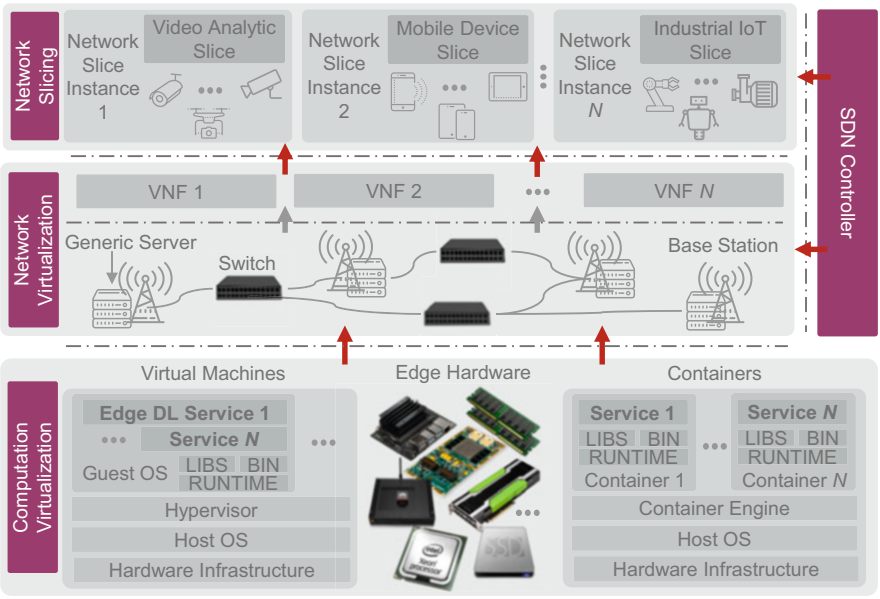

Fig. 2.3 Virtualizing edge computing infrastructure and networks

The combination of edge computing and AI to form high-performance Edge AI services requires the coordinated integration of computing, networking, and communication resources, as depicted in Fig. 2.3. Specifically, both the computation virtualization and the integration of network virtualization, and management technologies are necessary. In this section, we discuss potential virtualization technologies for the edge.

2.4.1 Virtualization Techniques

Currently, there are two main virtualization strategies: Virtual Machine (VM) and container. In general, VM is better at isolating while container provides easier deployment of repetitive tasks [42]. With VM virtualization at operating system level, a VM hypervisor splits a physical server into one or multiple VMs, and can easily manage each VM to execute tasks in isolation. Besides, the VM hypervisor can allocate and use idle computing resources more efficiently by creating a scalable system that includes multiple independent virtual computing devices. For example, VM-based cloudlets can provide end-users with more convenient edge computing services, and users can use VM technology to quickly instantiate a custom computing service on a nearby edge server, and then use this service to rapidly respond to the resource-intensive local task.

In contrast to VM, container virtualization is a more flexible tool for packaging, delivering, and orchestrating software infrastructure services and applications. Container virtualization for edge computing can effectively reduce the workload execution time with high-performance and storage requirements, and can also deploy a large number of services in a scalable and straightforward fashion [43]. A container consists of a single file that includes an application and execution environment with all dependencies, which makes it enable efficient service handoff to cope with user mobility [44]. Owning to that the execution of applications in the container does not depend on additional virtualization layers as in VM virtualization, the processor consumption and the amount of memory required to execute the application are significantly reduced.

2.4.2 Network Virtualization

Traditional networking functions, combined with specific hardware, is not flexible enough to manage edge computing networks in an on-demand fashion. In order to consolidate network device functions into industry-standard servers, switches, and storage, Network Functions Virtualization (NFV) enables Virtual Network Functions (VNFs) to run in software, by separating network functions and services from dedicated network hardware. Further, Edge AI services typically require high bandwidth, low latency, and dynamic network configuration, while Software-defined Networking (SDN) allows rapid deployment of services, network programmability and multi-tenancy support, through three key innovations [45]: (1) Decoupling of control planes and data planes; (2) Centralized and programmable control planes; (3) Standardized application programming interface. With these advantages, it supports a highly customized network strategy that is well suited for the high bandwidth, dynamic nature of Edge AI services.

Network virtualization and edge computing benefit each other. On the one hand, NFV/SDN can enhance the interoperability of edge computing infrastructure. For example, with the support of NFV/SDN, edge nodes can be efficiently orchestrated and integrated with cloud data centers [46]. On the other hand, both VNFs and Edge AI services can be hosted on a lightweight NFV framework (deployed on the edge) [47], thus reusing the infrastructure and infrastructure management of NFV to the largest extent possible [48].

2.4.3 Network Slicing

Network slicing is a form of agile and virtual network architecture, a high-level abstraction of the network that allows multiple network instances to be created on top of a common shared physical infrastructure, each of which optimized for specific services. With increasingly diverse service and QoS requirements, network slicing,

implemented by NFV/SDN, is naturally compatible with distributed paradigms of edge computing. To meet these, network slicing can be coordinated with joint optimization of computing and communication resources in edge computing networks [49]. Figure 2.3 depicts an example of network slicing based on edge virtualization. In order to implement service customization in network slicing, virtualization technologies and SDN must be together to support tight coordination of resource allocation and service provision on edge nodes while allowing flexible service control. With network slicing, customized and optimized resources can be provided for Edge AI services, which can help reduce latency caused by access networks and support dense access to these services [50].

2.5 Value Scenarios for Edge Computing

Edge computing is a technology that is very relevant to actual industrial practice, so its development cannot be separated from the actual demand generated in industrial production and needs to rely on actual application scenarios. Therefore, we will introduce some of the important value scenarios in edge computing, including smart parks, video surveillance, and industrial internet of things.

2.5.1 Smart Parks

The smart park uses a new generation of AI and communication technologies to sense, monitor, analyze, control, and integrate resources at key links in the park, and then implements timely and appropriate responses to various needs. Therefore, the operation of the smart park has the ability of self-organization, self-operation, and self-optimization, which can create an efficient, convenient, and personalized development space for the enterprises in the park.

In the smart park scenario, edge computing is mainly responsible for the following functions. (1) Massive network connection and management. Network management and automated operation and maintenance based on software-defined networking (SDN). (2) Real-time data collection and processing. In smart park applications such as face recognition and security alarms, it is required to meet the requirements of rapid response with real-time data collection of terminal nodes and local processing or end-to-edge collaborative processing. (3) Local business autonomy. In applications such as buildings intelligent self-control, when external network connections are interrupted, it is required to make local business autonomy so that the local system continues to execute local business logic normally.

2.5.2 Video Surveillance

The development in the field of video surveillance is transitioning from "seeing clearly" to "understanding." Video surveillance generates certain storage and computing requirements, and some surveillance tasks have real-time requirements. Therefore, the network collaboration mechanism of edge computing can reduce the reliance on cloud computing to improve the storage and computing efficiency of video surveillance.

In video surveillance scenarios, edge computing is mainly responsible for the following functions. (1) Edge node image recognition and video analysis. (2) Edge node intelligent storage mechanism. We can optimize data storage strategies and increase storage utilization of edge nodes based on video analysis results. (3) Edge–cloud collaboration. We can train models on cloud, and deploy trained models at edge rapidly.

2.5.3 Industrial Internet of Things

The application scenarios of the Industrial Internet of Things are relatively complicated. Different industries have different requirements for the level of digitalization and intelligence of equipment, and the demands for edge computing are also quite different. However, the following problems are common in most industrial IoT application scenarios. (1) Data is heterogeneous from multiple sources and lacks a unified format. (2) Many on-site network protocols make interconnection difficult. (3) Insufficient security protection measures for key data in the industrial production process.

Edge computing can solve the problems faced in the Industrial Internet of Things scenario through the following methods. (1) End nodes unify the format of heterogeneous data through preprocessing. (2) A unified industrial field network based on OPC UA over TSN to achieve data interconnection and interoperability. (3) Edge computing security mechanisms and solutions adapted to manufacturing scenarios.

References

1. M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies, The case for VM-based cloudlets in mobile computing. IEEE Pervasive Comput. **8**(4), 14–23 (2009)
2. M. Aazam, E. Huh, Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT, in *Proceedings of the IEEE 29th International Conference on Advanced Information Networking and Applications (AINA 2019)* (2015), pp. 687–694
3. F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the Internet of Things, in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing* (2012), pp. 13–16

4. F. Bonomi, R. Milito, P. Natarajan, J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics* (Springer, Cham, 2014), pp. 169–186
5. Mobile-Edge Computing—Introductory Technical White Paper, ETSI. https://portal.etsi.org/Portals/0/TBpages/MEC/Docs/Mobile-edge_Computing-IntroductoryTechnicalWhitePaperV1%2018-09-14.pdf
6. Multi-access Edge Computing. http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing
7. K. Bilal, O. Khalid, A. Erbad, S.U. Khan, Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers. Comput. Netw. **130**(2018), 94–120 (2018)
8. Openfog reference architecture for fog computing. https://www.openfogconsortium.org/ra/
9. Y. Kang, J. Hauswald, C. Gao et al., Neurosurgeon: collaborative intelligence between the cloud and mobile edge, in *Proceedings of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017)* (2017), pp. 615–629
10. G. Li, L. Liu, X. Wang et al., Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge, in *Proceedings of the International Conference on Artificial Neural Networks (ICANN 2018)* (2018), pp. 402–411
11. Y. Huang, Y. Zhu, X. Fan et al., Task scheduling with optimized transmission time in collaborative cloud-edge learning, in *Proceedings of the 27th International Conference on Computer Communication and Networks (ICCCN 2018)* (2018), pp. 1–9
12. What is Azure Data Box Edge? https://docs.microsoft.com/zh-cn/azure/databox-online/data-box-edge-overview
13. An all-scenario AI infrastructure solution that bridges 'device, edge, and cloud' and delivers unrivaled compute power to lead you towards an AI-fueled future. https://e.huawei.com/en/solutions/business-needs/data-center/atlas
14. Snapdragon 8 Series Mobile Platforms. https://www.qualcomm.com/products/snapdragon-8-series-mobile-platforms
15. Kirin. http://www.hisilicon.com/en/Products/ProductList/Kirin
16. The World's First Full-Stack All-Scenario AI Chip. http://www.hisilicon.com/en/Products/ProductList/Ascend
17. MediaTek Helio P60. https://www.mediatek.com/products/smartphones/mediatek-helio-p60
18. NVIDIA Turing GPU Architecture. https://www.nvidia.com/en-us/geforce/turing/
19. N.P. Jouppi, A. Borchers, R. Boyle, P.L. Cantin, B. Nan, In-datacenter performance analysis of a tensor processing unit, in *Proceedings of the 44th International Symposium on Computer Architecture (ISCA 2017)* (2017), pp. 1–12
20. Y. Xiong, Y. Sun, L. Xing, Y. Huang, Extend cloud to edge with KubeEdge, in *Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)* (2018), pp. 373–377
21. OpenEdge, extend cloud computing, data and service seamlessly to edge devices. https://github.com/baidu/openedge
22. Azure IoT Edge, extend cloud intelligence and analytics to edge devices. https://github.com/Azure/iotedge
23. EdgeX, the Open Platform for the IoT Edge. https://www.edgexfoundry.org/
24. Akraino Edge Stack. https://www.lfedge.org/projects/akraino/
25. E. Nurvitadhi, G. Venkatesh, J. Sim et al., Can FPGAs beat GPUs in accelerating next-generation deep neural networks? in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017)* (2017), pp. 5–14
26. S. Jiang, D. He, C. Yang et al., Accelerating mobile applications at the network edge with software-programmable FPGAs, in *2018 IEEE Conference on Computer Communications (INFOCOM 2018)* (2018), pp. 55–62
27. A. Ignatov, R. Timofte, W. Chou et al., AI benchmark: running deep neural networks on android smartphones (2018). arXiv:1810.01109
28. D. Bernstein, Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput. **1**(3), 81–84 (2014)

29. Microsoft Cognitive Toolkit (CNTK), an open source deep-learning toolkit. https://github.com/microsoft/CNTK
30. S. Tokui, K. Oono et al., Chainer: a next-generation open source framework for deep learning, in *Proceedings of the Workshop on Machine Learning Systems (LearningSys) in the Twenty-Ninth Annual Conference on Neural Information Processing Systems (NeurIPS 2015)* (2015), pp. 1–6
31. M. Abadi, P. Barham et al., TensorFlow: a system for large-scale machine learning, in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI 2016)* (2016), pp. 265–283
32. Deeplearning4j: Open-source distributed deep learning for the JVM, Apache Software Foundation License 2.0. https://deeplearning4j.org
33. Deploy machine learning models on mobile and IoT devices. https://www.tensorflow.org/lite
34. T. Chen, M. Li, Y. Li et al., MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems (2015). arXiv:1512.01274
35. PyTorch: tensors and dynamic neural networks in Python with strong GPU acceleration. https://github.com/pytorch/
36. Core ML: Integrate machine learning models into your app. https://developer.apple.com/documentation/coreml?language=objc
37. NCNN is a high-performance neural network inference framework optimized for the mobile platform. https://github.com/Tencent/ncnn
38. MNN is a lightweight deep neural network inference engine. https://github.com/alibaba/MNN
39. Multi-platform embedded deep learning framework. https://github.com/PaddlePaddle/paddle-mobile
40. MACE is a deep learning inference framework optimized for mobile heterogeneous computing platforms. https://github.com/XiaoMi/mace
41. X. Wang, M. Magno, L. Cavigelli, L. Benini, FANN-on-MCU: an open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things (2019). arXiv:1911.03314
42. Z. Tao, Q. Xia, Z. Hao, C. Li, L. Ma, S. Yi, Q. Li, A survey of virtual machine management in edge computing. Proc. IEEE **107**(8), 1482–1499 (2019)
43. R. Morabito, Virtualization on internet of things edge devices with container technologies: a performance evaluation. IEEE Access **5**, 8835–8850 (2017)
44. L. Ma, S. Yi, N. Carter, Q. Li, Efficient live migration of edge services leveraging container layered storage. IEEE Trans. Mob. Comput. **18**(9), 2020–2033 (2019)
45. A. Wang, Z. Zha, Y. Guo, S. Chen, Software-defined networking enhanced edge computing: a network-centric survey. Proc. IEEE **107**(8), 1500–1519 (2019)
46. Y.D. Lin, C.C. Wang, C.Y. Huang, Y. C. Lai, Hierarchical CORD for NFV datacenters: resource allocation with cost-latency tradeoff. IEEE Netw. **32**(5), 124–130 (2018)
47. L. Li, K. Ota, M. Dong, DeepNFV: A lightweight framework for intelligent edge network functions virtualization. IEEE Netw. **33**(1), 136–141 (2019)
48. Mobile Edge Computing A key technology towards 5G, ETSI. https://www.etsi.org/images/files/ETSIWhitePapers/etsiwp11mecakeytechnologytowards5g.pdf
49. H.-T. Chien, Y.-D. Lin, C.-L. Lai, C.-T. Wang, End-to-end slicing as a service with computing and communication resource allocation for multi-tenant 5G systems. IEEE Wirel. Commun. **26**(5), 104–112 (2019)
50. T. Taleb, K. Samdanis, B. Mada, H. Flinck, S. Dutta, D. Sabella, On multi-access edge computing: a survey of the emerging 5g network edge cloud architecture and orchestration. IEEE Commun. Surv. Tutorials **19**(3), 1657–1681 (2017)

Chapter 3
Fundamentals of Artificial Intelligence

Abstract AI is a broad field of research that includes many methods of research value. However, due to the characteristics of edge computing in its operating structure and computing resources, deep learning has become the most closely related and representative method in AI for edge computing. In addition, due to the limitations of resources in edge computing, there is a lack of targeted solutions for resource-intensive deep learning. Therefore, in this book we focus on deep learning that require high computing resources. With respect to CV, NLP, and AI, DL is adopted in a myriad of applications and corroborates its superior performance by LeCun et al. (Nature 521(7553):436–444, 2015). Currently, a large number of GPUs, TPUs, or FPGAs are required to be deployed in the cloud to process DL service requests.Through the introduction of the previous two chapters, this book has made an in-depth analysis of the development bottlenecks of the current cloud computing model. The reader can understand that the current response time requirements of some deep learning applications are extremely demanding, and cloud computing can no longer meet the needs. Therefore, it is necessary to consider transferring the task of deep learning to the edge computing framework. Nonetheless, the edge computing architecture, on account of it covers a large number of distributed edge devices, can be utilized to better serve DL. Certainly, edge devices typically have limited computing power or power consumption compared to the cloud. Therefore, the combination of DL and edge computing is not straightforward and requires a comprehensive understanding of DL models and edge computing features for design and deployment. In this section, we compendiously introduce DL and related technical terms, paving the way for discussing the integration of DL and edge computing.

3.1 Artificial Intelligence and Deep Learning

Artificial intelligence (AI) was first proposed at the Dartmouth Conference in 1956. Early leaders in this field had the vision that intelligent machines like humans would appear in the near future. In a narrow sense, AI is a machine that can well

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_3

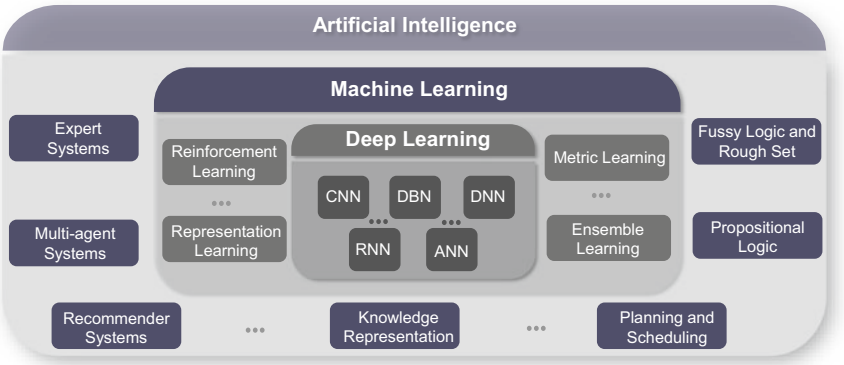

Fig. 3.1 The relationship between artificial intelligence, machine learning, and deep learning

demonstrate some of the human functions, such as image recognition or speech recognition applications [30].

Machine learning is one way to implement AI. People can train machine learning algorithms to make machines have the ability to learn and reason. For example, to determine whether there is a cat in a picture: First, a large number of labeled pictures are input into the machine, and the machine learning algorithm trains model parameters based on these picture data, and finally obtains a model that can accurately determine whether a cat is in the picture.

Deep learning and reinforcement learning are both types of machine learning. Deep learning is suitable for the processing of huge amounts of data. It is inspired by the structure and function of the human brain, that is, the neurons connected by layers extract data features to complete learning. Reinforcement learning is more suitable for giving machines stronger self-decision capabilities.

As shown in Fig. 3.1, AI, machine learning and deep learning are as follows. (1) AI is a relatively broad research area which focuses on researching solutions similar to human intelligence. (2) Machine learning is an important practical way of AI which can make machines have the ability to learn in the process of interaction with the environment. (3) Deep learning, as a subset of machine learning, can use neural networks to mimic the connectivity of the human brain to classify datasets and discover correlations between them.

The fusion of edge computing and deep learning can accelerate artificial intelligence technology into humans' lives. In [1], researchers optimize network performance and protect user data privacy by deploying deep learning combined with edge computing in IoT application scenarios.

Although the only representative technology on the macro-level of artificial intelligence may not be deep learning, the most representative of artificial intelligence technology on the edge is deep learning, because the edge side already can support

machine learning models with less computing power requirements. Deep learning technology with large power requirements also needs a set of targeted solutions. In this book, we focus on deep learning technologies with high computing power requirements.

3.2 Neural Networks in Deep Learning

DL models consist of various types of Deep Neural Networks (DNNs) [2].The most basic neural network architecture is composed of an input layer, a hidden layer, and an output layer. The so-called deep neural network refers to a sufficient number of hidden layers between the input layer and the output layer. Such deep learning is composed of various neural networks. Fundamentals of DNNs in terms of basic structures and functions are introduced as follows.

3.2.1 Fully Connected Neural Network (FCNN)

The output of each layer of FCNN, i.e., Multi-Layer Perceptron (MLP), is fed forward to the next layer, as in Fig. 3.2. Between contiguous FCNN layers, the output of a neuron (cell), either the input or hidden cell, is directly passed to and activated by neurons belong to the next layer [3]. FCNN can be used for feature extraction and function approximation, however, with high complexity, modest performance, and slow convergence.

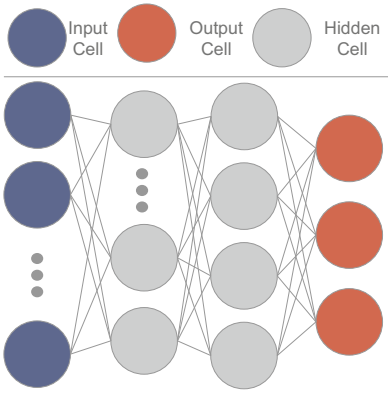

Fig. 3.2 Fully connected neural network

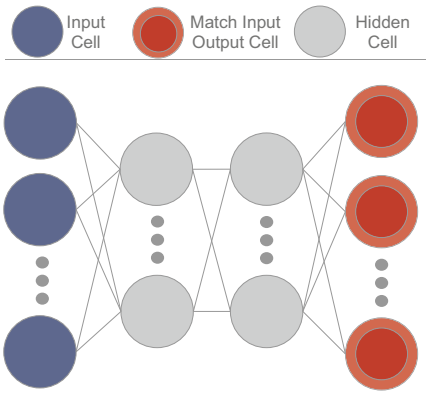

Fig. 3.3 Auto-encoder

3.2.2 Auto-Encoder (AE)

AE, as in Fig. 3.3, is actually a stack of two NNs that replicate input to its output in an unsupervised learning style. The first NN learns the representative characteristics of the input (encoding). The second NN takes these features as input and restores the approximation of the original input at the match input output cell, used to converge on the identity function from input to output, as the final output (decoding).

The identity function does not seem to be meaningful for learning, but if we add certain restrictions to the auto-encoder neural network, such as limiting the number of neurons in the hidden layer, then the entire process can be described as first reducing the dimension and then reconstructing it. Also, it is necessary to ensure that the reduced-dimensional data of the encoder can save relatively complete information of the original data so that the decoder can be reconstructed. This results in a very interesting structure. Since AEs are able to learn the low-dimensional useful features of input data to recover input data, it is often used to classify and store high-dimensional data [4].

3.2.3 Convolutional Neural Network (CNN)

By employing pooling operations and a set of distinct moving filters, CNNs seize correlations between adjacent data pieces, and then generate a successively higher level abstraction of the input data, as in Fig. 3.4. Compared to FCNNs, the core idea of CNNs is convolutional layers and pooling layers:

(1) Convolutional layers: In short, it is to extract image features. Convolutional layers are similar to what we call "filters" on computer image processing. The convolution operation of the original image with the convolution layer can enhance some features in the original information and reduce noise. At the

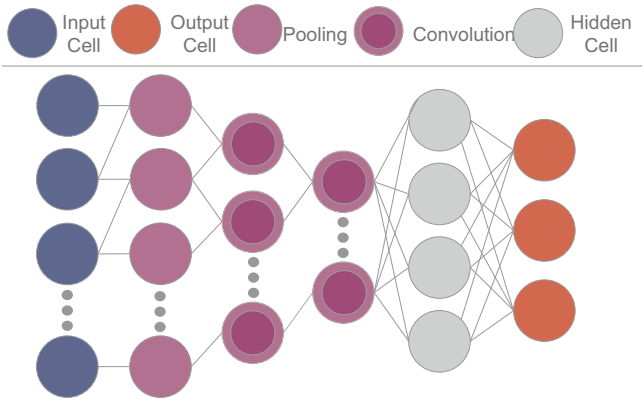

Fig. 3.4 Convolutional neural network

same time, the principle of "weight sharing" greatly reduces the parameters to be solved.

(2) Pooling layers: After the operation of the convolutional layer, we get the feature map we want. The pooling layer compresses the feature map. On the one hand, the feature map is made smaller, simplifying the computational complexity of the network; on the other hand, feature extraction and compression can be performed to extract the main features.

So it can extract features while reducing the model complexity, which mitigates the risk of overfitting [5]. These characteristics make CNNs achieve remarkable performance in image processing and also useful in processing structural data similar to images.

3.2.4 Generative Adversarial Network (GAN)

GAN originates from game theory. As illustrated in Fig. 3.5, GAN is composed of *generator* and *discriminator*. The goal of the *generator* is to learn about the true data distribution as much as possible by deliberately introducing feedback at the back-fed input cell, while the *discriminator* is to correctly determine whether the input data is coming from the true data or the *generator*. These two participants need to constantly optimize their ability to generate and distinguish in the adversarial process until finding a Nash equilibrium [6].

The basic principle of GAN can be simply described as follows:

At the beginning, we have random variables Z and real data T; we input random data Z into the *generator*, and the output $G(Z)$ of the *generator* network training should follow the distribution of real data T as much as possible. And then, the $G(Z)$ and T input *discriminator* are used for normal discrimination. The data from the true sample T is labeled as 1; the fake sample data $G(Z)$ is labeled as 0. The

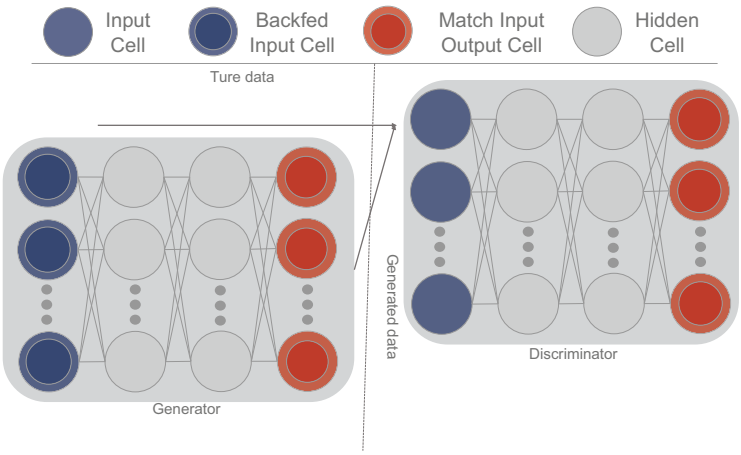

Fig. 3.5 Generative adversarial network

goal of the *generator* is to make the fake data *G(Z)* and the real data *T* generated in the *discriminator* behave in the same status, to achieve the purpose of the false real disorder. These two processes that oppose each other and iteratively optimize the performance of the generating network and the discriminating network.

According to the features learned from the real information, a well-trained *generator* can thus fabricate indistinguishable information.

3.2.5 Recurrent Neural Network (RNN)

RNNs are designed for handling sequential data. As depicted in Fig. 3.6, each neuron in RNNs not only receives information from the upper layer but also receives information from the previous channel of its own [7]. In general, RNNs are natural choices for predicting future information or restoring missing parts of sequential data. However, a serious problem with RNNs is the gradient explosion. LSTM, as in Fig. 3.7, improving RNN with adding a gate structure and a well-defined memory cell, can overcome this issue by controlling (prohibiting or allowing) the flow of information [8]. A common LSTM is composed of three gates: input gate, forget gate, and output gate. The input gate is used to select the content update of the memory cell, the forget gate determines which information needs to be discarded, and the output gate controls which part of the memory cell will be output at this moment. According to these characteristics, LSTM is widely used in NLP fields such as speech recognition and machine translation.

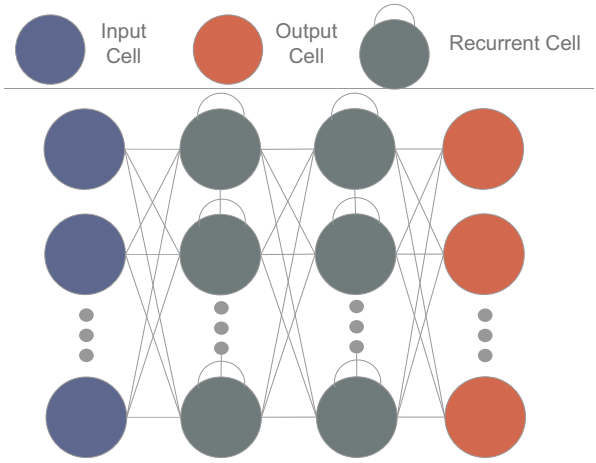

Fig. 3.6 Recurrent neural network

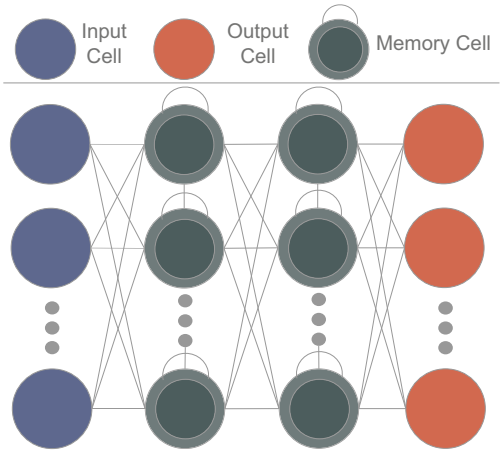

Fig. 3.7 Long short-term memory

3.2.6 Transfer Learning (TL)

The five deep neural network models and their derivatives introduced earlier have been developed and widely used in real life for a long time, and have pushed artificial intelligence technology to the climax again and again. However, behind these algorithms, there is a huge demand for the computing power of the machine. At the same time, some example applications including speech recognition, image processing, also require a large amount of labeled data. Leaving aside the later model training, just labeling the data is a tedious and costly task. So can the trained network be improved to solve some common problems? For example, we currently have a network that can identify bicycles in images. Can we train a network that can identify motorcycles based on this network, instead of starting from scratch? This can greatly reduce development costs.

TL can transfer knowledge, as shown in Fig. 3.8, from the source domain to the target domain so as to achieve better learning performance in the target domain [9]. By using TL, existing knowledge learned by a large number of computation resources can be transferred to a new scenario, and thus accelerating the training process and reducing model development costs. However, there is a problem with transfer learning. It is often applied to some small and stable datasets, making it difficult to get a wider application.

Recently, a novel form of TL emerges, viz., Knowledge Distillation (KD) [10] emerges. As indicated in Fig. 3.9, KD can extract implicit knowledge from a well-trained model (teacher), inference of which possess excellent performance but requires high overhead. Then, by designing the structure and objective function of the target DL model, the knowledge is "transferred" to a smaller DL model

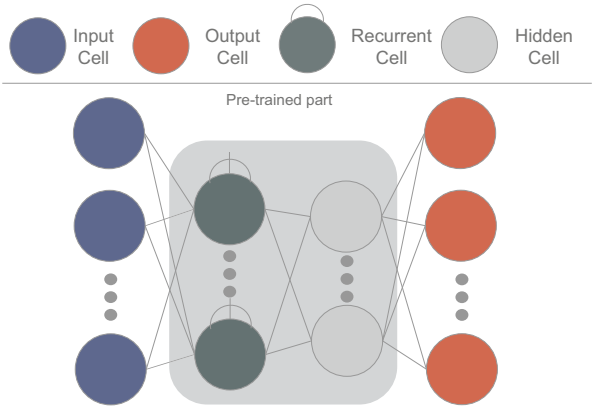

Fig. 3.8 Transfer learning

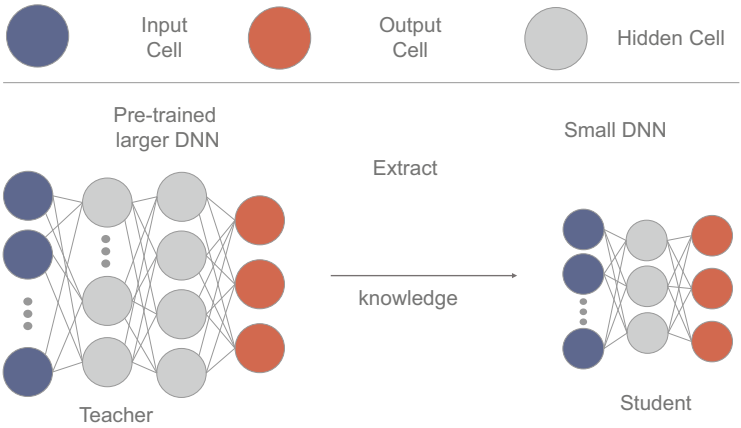

Fig. 3.9 Knowledge distillation

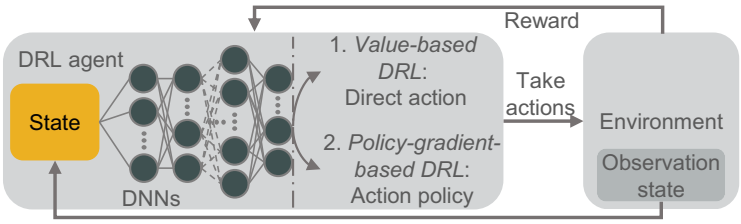

Fig. 3.10 Value-based and policy-gradient-based DRL approaches

(student), so that the significantly reduced (pruned or quantized) target DL model achieves high performance as possible.

3.3 Deep Reinforcement Learning (DRL)

As depicted in Fig. 3.10, the goal of RL is to enable an agent in the environment to take the best action in the current state to maximize long-term gains, where the interaction between the agent's action and state through the environment is modeled as a Markov Decision Process (MDP). DRL is the combination of DL and RL, but it focuses more on RL and aims to solve decision-making problems. The role of DL is to use the powerful representation ability of DNNs to fit the value function or the direct strategy to solve the explosion of state-action space or continuous state-action space problem. By virtue of these characteristics, DRL becomes a powerful solution in robotics, finance, recommendation system, wireless communication, etc. [11].

3.3.1 Reinforcement Learning (RL)

The Markov decision process modeling the RL task can be described formally as a quaternion $E = (S, A, P, R)$ for reinforcement learning tasks, where:

(1) E is the environment in which the agent is located;
(2) S is the set of all environmental states, where each state $s \in S$ represents the environmental description perceived by the agent;
(3) A is the set of actions that the agent can perform and $a \in A$ acts on the current state s;
(4) P is a state transition distribution function, that is, after acting a at a certain time, P will cause the current environment to transition to another state with a certain probability;
(5) R is the reward function, which represents the reward value obtained by the agent after acting in state s.

Reinforcement learning adjusts the decision-making process through the reward value feedback from the reward function R. Therefore, a sequence is formed during the constant interaction between the agent and the environment, and this process is a sequence decision process. The Markov decision process is the formulation of this process and provides a convenient and practical means.

3.3.2 Value-Based DRL

As a representative of value-based DRL, Deep Q-Learning (DQL) uses DNNs to fit action values, successfully mapping high-dimensional input data to actions [12]. In order to ensure stable convergence of training, experience replay method is adopted to break the correlation between transition information and a separate target network is set up to suppress instability. DQN is the first to combine deep learning models with reinforcement learning to successfully learn control strategies directly from high-dimensional inputs. However, DQN still has many shortcomings. In order to solve these problems, DQN has derived many improved versions.

Double Deep Q-Learning (Double-DQL) can deal with that DQL generally overestimating action values [13], and Dueling Deep Q-Learning (Dueling-DQL) [14] can learn which states are (or are not) valuable without having to learn the effect of each action at each state.

3.3.3 Policy-Gradient-Based DRL

Policy gradient is another common strategy optimization method, such as Deep Deterministic Policy Gradient (DDPG) [15], Asynchronous Advantage Actor-Critic (A3C) [16], Proximate Policy Optimization (PPO) [17], etc. It updates the policy parameters by continuously calculating the gradient of the policy expectation reward with respect to them, and finally converges to the optimal strategy [18]. Therefore, when solving the DRL problem, DNNs can be used to parameterize the policy, and then be optimized by the policy gradient method. Further, Actor-Critic (AC) framework is widely adopted in policy-gradient-based DRL, in which the policy DNN is used to update the policy, corresponding to the Actor; the value DNN is used to approximate the value function of the state-action pair, and provides gradient information, corresponding to the Critic.

3.4 Distributed DL Training

At present, training DL models in a centralized manner consumes a lot of time and computation resources, hindering further improving the algorithm performance. Nonetheless, distributed training can facilitate the training process by taking full

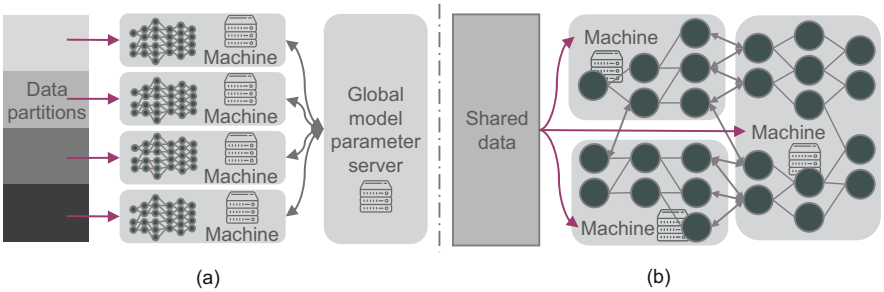

Fig. 3.11 Distributed training in terms of data and model parallelism. (**a**) Data parallelism. (**b**) Model parallelism

advantage of parallel servers. There are two common ways to perform distributed training, i.e., *data parallelism* and *model parallelism* [19–22] as illustrated in Fig. 3.11.

Model parallelism first splits a large DL model into multiple parts and then feeds data samples for training these segmented models in parallel. This not only can improve the training speed but also deal with the circumstance that the model is larger than the device memory. Training a large DL model generally requires a lot of computation resources, even thousands of CPUs are required to train a large-scale DL model. In order to solve this problem, distributed GPUs can be utilized for model parallel training [23]. Data parallelism means dividing data into multiple partitions, and then, respectively, training copies of the model in parallel with their own allocated data samples. By this means, the training efficiency of model training can be improved [24].

Coincidentally, a large number of end devices, edge nodes, and cloud data centers, are scattered and envisioned to be connected by virtue of edge computing networks. These distributed devices can potentially be powerful contributors once the DL training jumps out of the cloud.

3.4.1 Data Parallelism

If the working nodes do not have public memory and the training data is large, we need to divide the dataset and allocate it to each working node. There are two classic methods for dividing data samples:

(1) Based on the "random sampling" method: random sampling can help local training data on each machine to be independent and identically distributed with the original training data, but if the number of training data types is very large, there may always be training samples chosen.

(2) A method based on "scramble and segmentation": randomly scramble training data divide it into corresponding small parts according to the number of working nodes, and then allocate the divided data to the working nodes.

On the other hand, data parallelism in deep learning training can also be considered as a parallelization of gradient descent. When using the sample data to train the model, the model will constantly adjust the parameters to bring the predicted values closer to the actual values. For this purpose, a function is usually defined to describe the difference between the predicted value and the actual value, called loss function. Thus, the training of the model can be transformed into a problem that minimizes the loss function. To solve this problem, a gradient descent method is usually used, that is, for a given parameter and loss function, the algorithm minimizes the loss function by updating the parameters in the opposite direction of the gradient. For example, Stochastic Gradient Descent (SGD) is a representative of this kind of method, which first assigns a set of initial values to the parameters, and then randomly selects a data from the training sample set and updates the parameters with it in each iteration, and finally ends when the convergence state is reached.

But for a single computing device that cannot accommodate a larger model, the model parallel mode is a good choice.

3.4.2 Model Parallelism

In deep neural networks, the dependencies between parameters are serious, but the hierarchical nature of neural networks has brought some convenience to model parallelism. For larger neural network models, it means that a large number of network layers are arranged in the direction from the input layer to the output layer. A natural and easy to implement a method is that each working node undertakes one or more of the computing tasks in it. For computers with different computing power, we assign tasks that match their computing power as close as possible. Here, we can use distributed GPUs for model parallel training [23].

3.5 Potential DL Libraries for Edge

Development and deployment of DL models rely on the support of various DL libraries. However, different DL libraries have their own application scenarios. For deploying DL on and for the edge, efficient lightweight DL libraries are required. Features of DL frameworks potentially supporting future edge intelligence are listed in Table 2.4 (excluding libraries unavailable for edge devices, such as Theano [25]).

As the heat of deep learning continues to rise, various deep learning libraries are also emerging, which provides a convenient way for the application and research of

deep learning. In this section, we have selected several types of mainstream deep learning libraries for a brief introduction.

(1) TensorFlow [26]. TensorFlow is a relatively high-level framework that can be easily used to design neural network structures without having to write C++ or CUDA code for the most efficient implementation. Compared to other frameworks, another important feature of TensorFlow is its flexible portability. With little modification, the same code can be easily deployed to a PC, server or mobile device with any number of CPUs or GPUs. Besides, TensorFlow is now preliminarily available for dynamic computation graph.
(2) Caffe (Convolutional Architecture for Fast Feature Embedding) [27]. The core concept of Caffe is the layer, making input data and perform calculations inside the DL model possible. It is widely used in computer vision such as face recognition, image classification, and target tracking.
(3) Theano [25]. The core of Theano is a mathematical expression compiler designed to handle large-scale neural network calculations. It compiles various user-defined calculations into efficient low-level codes. However, deploying Theano models is inconvenient and does not support a variety of mobile devices, therefore lacking applications in production environments.
(4) Keras [28]is a high-level neural network library. Keras is written in pure Python and is based on TensorFlow and Theano. Keras understands deep learning models as an independent sequence or graph, which is convenient for users to combine configurable modules together with minimal code. Form a completed deep learning model. Compared with Caffe, Keras does not have a separate model configuration file, but is entirely composed of python, which is convenient for users to debug.
(5) (Py)Torch [29] and Chainer. (Py)Torch, Chainer support dynamic computation graph, i.e., constructing a graph for each line of code as part of a complete computational graph. Even if the calculation diagram is not completely built, small calculation diagrams, as independent components, can also be performed. This feature is attractive to researchers and engineers engaged in time series data analysis and natural language processing. To be noted, Caffe [27] has been merged into (Py)Torch.

References

1. H. Li, K. Ota, M. Dong, Learning IoT in edge: deep learning for the Internet of Things with edge computing. IEEE Netw. **32**(1), 96–101 (2018)
2. S.S. Haykin, K. Elektroingenieur, *Neural Networks and Learning Machines* (Pearson Prentice Hall, Englewood Cliffs, 2009)
3. R. Collobert, S. Bengio, Links between perceptrons, MLPs and SVMs, in *Proceeding of the Twenty-first International Conference on Machine Learning (ICML 2004)* (2004), p. 23
4. C.D. Manning, C.D. Manning, H. Schütze, *Foundations of Statistical Natural Language Processing* (MIT Press, New York, 1999)

5. M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in *2014 European Conference on Computer Vision (ECCV 2014)* (2014), pp. 818–833
6. I. Goodfellow, J. Pouget-Abadie, M. Mirza, et al., Generative adversarial nets, in *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)* (2014), pp. 2672–2680
7. J. Schmidhuber, Deep learning in neural networks: an overview. Neural Netw. **61**, 85–117 (2015)
8. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
9. S.J. Pan, Q. Yang, A survey on transfer learning. IEEE Trans. Knowl. Data Eng. **22**(10), 1345–1359 (2010)
10. G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network (2015). arXiv preprint:1503.02531
11. S.S. Mousavi, M. Schukat, E. Howley, Deep reinforcement learning: an overview, in *Proceeding of the 2016 SAI Intelligent Systems Conference (IntelliSys 2016)* (2016), pp. 426–440
12. V. Mnih, K. Kavukcuoglu, D. Silver, et al., Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
13. H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in *Proceeding of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)* (2016), pp. 2094–2100
14. Z. Wang, T. Schaul, M. Hessel, et al., Dueling network architectures for deep reinforcement learning, in *Proceeding of the 33rd International Conference on Machine Learning (ICML 2016)* (2016), pp. 1995–2003
15. T.P. Lillicrap, J.J. Hunt, A. Pritzel, et al., Continuous control with deep reinforcement learning, in *Proceeding of the 6th International Conference on Learning Representations (ICLR 2016)* (2016)
16. V. Mnih, A.P. Badia, M. Mirza, et al., Asynchronous methods for deep reinforcement learning, in *Proceeding of the 33rd International Conference on Machine Learning (ICML 2016)* (2016), pp. 1928–1937
17. J. Schulman, F. Wolski, P. Dhariwal, et al., Proximal policy optimization algorithms (2017). arXiv preprint:1707.06347
18. R.S. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in *Proceeding of the 12th International Conference on Neural Information Processing Systems (NeurIPS 1999)* (1999), pp. 1057–1063
19. A.S. Monin, A.M. Yaglom, Large scale distributed deep networks, in *Proceeding of Advances in Neural Information Processing Systems 25 (NeurIPS 2012)* (2012), pp. 1223–1231
20. Y. Zou, X. Jin, Y. Li, et al., Mariana: Tencent deep learning platform and its applications. *Proc. VLDB Endow.* **7**(13), 1772–1777 (2014)
21. X. Chen, A. Eversole, G. Li, et al., Pipelined back-propagation for context-dependent deep neural networks, in *Proceeding of 13th Annual Conference of the International Speech Communication Association (INTERSPEECH 2012)* (2012), pp. 26–29
22. M. Stevenson, R. Winter, et al., 1-Bit stochastic gradient descent and its application to data-parallel distributed training of speech DNNs, in *15th Annual Conference of the International Speech Communication Association (INTERSPEECH 2014)* (2014), pp. 1058–1062
23. A. Coates, B. Huval, T. Wang, et al., Deep learning with cots HPC systems, in *Proceeding of the 30th International Conference on Machine Learning (PMLR 2013)* (2013), pp. 1337–1345
24. P. Moritz, R. Nishihara, I. Stoica, and M. I. Jordan, SparkNet: Training Deep Networks in Spark. arXiv preprint:1511.06051, 2015.
25. *Theano is a Python Library that Allows you to Define, Optimize, and Evaluate Mathematical Expressions Involving Multi-Dimensional Arrays Efficiently*. https://github.com/Theano/Theano
26. M. Abadi, P. Barham, et al., TensorFlow: a system for large-scale machine learning, in *Proceeding of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI 2016)* (2016), pp. 265–283

27. Y. Jia, E. Shelhamer, et al., Caffe: convolutional architecture for fast feature embedding, in *Proceedings of the 22nd ACM international conference on Multimedia* (2014), pp. 675–678
28. Géron, Aurélien, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (O'Reilly Media, Sebastopol, 2019)
29. A. Paszke, S. Gross, et al., PyTorch: an imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems* (2019), pp. 8024–8035
30. Y. LeCun, Y. Bengio, G. Hinton, Deep learning. Nature **521**(7553), 436–444 (2015)

Part II
Artificial Intelligence and Edge Computing

Chapter 4
Artificial Intelligence Applications on Edge

Abstract In general, AI services are currently deployed in cloud data centers (the cloud) for handling requests, due to the fact that most AI models are complex and hard to compute their inference results on the side of resource-limited devices. However, such kind of "end–cloud" architecture cannot meet the needs of real-time AI services such as real-time analytics, smart manufacturing, etc. Thus, deploying AI applications on the edge can broaden the application scenarios of AI especially with respect to the low-latency characteristic. In the following, we present edge AI applications and highlight their advantages over the comparing architectures without edge computing.

4.1 Real-time Video Analytic

Real-time video analytic is important in various fields, such as automatic pilot, VR and Augmented Reality (AR), smart surveillance, etc. In general, applying DL for it requires high computation and storage resources. Unfortunately, executing these tasks in the cloud often incurs high-bandwidth consumption, unexpected latency, and reliability issues. With the development of edge computing, those problems tend to be addressed by moving video analysis near to the data source, viz., end devices or edge nodes, as the complementary of the cloud. In this section, as depicted in Fig. 4.1, we summarize related works as a hybrid hierarchical architecture, which is divided into three levels: end, edge, and cloud.

4.1.1 Machine Learning Solution

Video analysis has various application types, such as face recognition, object recognition, trajectory tracking, etc. Meanwhile, the effects of different machine learning models on different video analysis tasks are also different. So more and more machine learning algorithms, such as principal component analysis, histogram analysis, artificial Neural networks, Bayesian classification, adaptive boosting

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_4

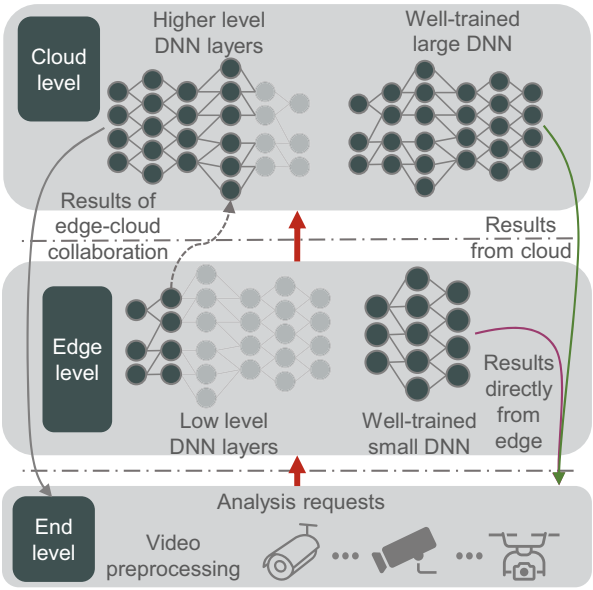

Fig. 4.1 The End–Edge–Cloud collaboration for performing real-time video analytic by AI

learning, etc., have been cited in the field of video analysis and computer vision and perform well in a certain range relatively. However, because the performance of traditional machine learning models is affected by the characteristics of the identified items (such as the shape, the color distribution, the relative position, etc.), it is difficult to use a single model to analyze the general video without noise reduction and feature extraction. Therefore, we prefer to combine several machine learning models to achieve higher system performance.

4.1.2 Deep Learning Solution

Let us take the face recognition as an example. Face recognition generally includes steps such as face detection, gender and age classification, face tracking, and feature matching. For different steps, we often choose different machine learning models. To solve the task of face detection, AdaBoost classifier, described in paper [1], is utilized. Detected fragments are preprocessed to align their luminance characteristics and to transform them to a uniform scale. After we select and process the image fragments, we will classify it by gender and age. The proposed gender and age classifiers are based on nonlinear SVM (Support Vector Machines) classifier with RBF kernel. In face tracking, an algorithm, proposed by B. Lucas and T. Kanade in paper [2], was chosen as the basic approach to solve the problem of

optical flow calculation, which can be defined as a two-dimensional projection of objects motion on an image plane, representing object pixel's trajectories.

4.1.2.1 End Level

At the end level, video capture devices such as smartphones and surveillance cameras are responsible for video capture, media data compression [3], image preprocessing, and image segmentation [4]. By coordinating with these participated devices, collaboratively training a domain-aware adaptation model can lead to better object recognition accuracy when used together with a domain-constrained deep model [5]. Besides, in order to appropriately offload the DL computation to the end devices, the edge nodes or the cloud, end devices should comprehensively consider trade-offs between video compression and key metrics, e.g., network condition, data usage, battery consumption, processing delay, frame rate and accuracy of analytics, and thus determine the optimal offloading strategy [3].

If various DL tasks are executed at the end level independently, enabling parallel analytics requires a solution that supports efficient multi-tenant DL. With the model pruning and recovery scheme, *NestDNN* [6] transforms the DL model into a set of descendant models, in which the descendant model with fewer resource requirements shares its model parameters with the descendant model requiring more resources, making itself nested inside the descendent model requiring more resources without taking extra memory space. In this way, the multi-capacity model provides variable resource-accuracy trade-offs with a compact memory footprint, hence ensuring efficient multi-tenant DL at the end level.

4.1.2.2 Edge Level

Numerous distributed edge nodes at the edge level generally cooperate with each other to provide better services. For example, *LAVEA* [7] attaches edge nodes to the same access point or BS as well as the end devices, which ensure that services can be as ubiquitous as Internet access. In addition, compressing the DL model on the edge can improve holistic performance. The resource consumption of the edge layer can be greatly reduced while ensuring the analysis performance, by reducing the unnecessary filters in CNN layers [8]. Besides, in order to optimize performance and efficiency, [9] presents an edge service framework, i.e., *EdgeEye*, which realizes a high-level abstraction of real-time video analytic functions based on DL. To fully exploit the bond function of the edge, *VideoEdge* [10] implements an end–edge–cloud hierarchical architecture to help achieve load balancing concerning analytical tasks while maintaining high analysis accuracy.

4.1.2.3 Cloud Level

At the cloud level, the cloud is responsible for the integration of DL models among the edge layer and updating parameters of distributed DL models on edge nodes [3]. Since the distributed model training performance on an edge node may be significantly impaired due to its local knowledge, the cloud needs to integrate different well-trained DL models to achieve global knowledge. When the edge is unable to provide the service confidently (e.g., detecting objects with low confidence), the cloud can use its powerful computing power and global knowledge for further processing and assist the edge nodes to update DL models.

4.2 Autonomous Internet of Vehicles (IoVs)

It is envisioned that vehicles can be connected to improve safety, enhance efficiency, reduce accidents, and decrease traffic congestion in transportation systems [11]. There are many information and communication technologies such as networking, caching, edge computing which can be used for facilitating the IoVs, though usually studied, respectively. On one hand, edge computing provides low-latency, high-speed communication, and fast-response services for vehicles, making automatic driving possible. On the other hand, DL techniques are important in various smart vehicle applications. Further, they are expected to optimize complex IoVs systems.

In [11], a framework which integrates these technologies is proposed. This integrated framework enables dynamic orchestration of networking, caching, and computation resources to meet requirements of different vehicular applications [11]. Since this system involves multi-dimensional control, a DRL-based approach is first utilized to solve the optimization problem for enhancing the holistic system performance. Similarly, DRL is also used in [12] to obtain the optimal task offloading policy in vehicular edge computing. Besides, Vehicle-to-Vehicle (V2V) communication technology can be taken advantaged to further connect vehicles, either as an edge node or an end device managed by DRL-based control policies [13]. In this section, the architecture of Autonomous Vehicle for edge computing can be showed in Fig. 4.2.

4.2.1 Machine Learning Solution

A car with autonomous driving is equipped with a variety of sensors, and a large amount of data generated by these sensors is also required to be processed promptly. One of the options for extracting information from this raw data and making decisions is to input this data into a machine learning system. Machine learning algorithms deployed on self-driving cars extract features from raw data to determine real-time road conditions and make reasonable decisions based on them.

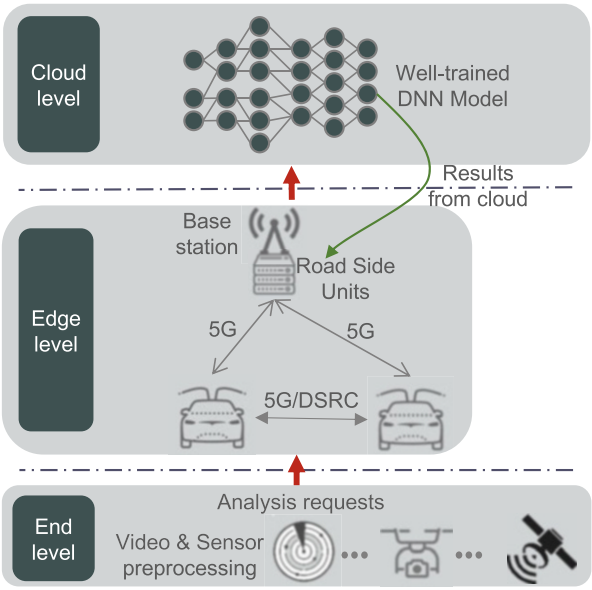

Fig. 4.2 Autonomous IoVs by edge AI

For example, in [14], the position of the human body was identified by LIDAR, and a high accuracy rate was achieved.

4.2.2 Deep Learning Solution

To drive safely, when driving a car, humans will not only consider road pedestrian factors, but also consider whether the road is smooth, traffic lights, driving conditions of front and rear vehicles, and so on. It is not enough to rely on machine learning systems for such heavy tasks. In [15], the algorithm subsystem in autonomous driving is composed of sensing, perception, and decision. The autonomous driving system obtains environmental data through sensing, and then uses deep learning algorithms to process the data and finally make a decision.

4.2.2.1 End Level

In terms of IoV, the end level data acquisition equipment mainly includes sensors and cameras, where the sensors can collect physical properties of vehicles, such as driving speed, acceleration, direction, distance, etc., and the camera mainly collects the surrounding image information of the vehicle, such as lanes, stop lines, etc. End devices generally do not undertake computing tasks and are mainly responsible for collecting and organizing all data to meet the computing needs of the edge level

and the cloud level. For example, the determination of the driving state of a vehicle is limited by the data from multiple dimensions, and we can all complete these tasks directly at the end level. Unlike the video analysis in Sect. 4.1, IoV pays more attention to the utility of delay on the performance of the overall system. Therefore, we hope that more computing tasks can be completed at the edge level, which requires that the end should be closer to the design requirements of the edge device.

4.2.2.2 Edge Level

As we mentioned earlier, lower latency means higher security in terms of IoV, so security is one of the most valued aspects of IoV. Compared with the cloud, edge devices have a greater advantage in transmission delay because closer to the end. This shows that we need to deploy more computing tasks to process at the edge layer, even when the communication between the edge and the cloud is cut off due to some special circumstances, the edge device can still maintain the normal operation of IoV for a short time. Therefore, we should make greater use of the computing and storage resources of the edge devices, and reserve a certain amount of spare resources in the edge level to avoid the circumstance that the edge devices cannot satisfy the minimum computing requirements of the entire system when communication between the cloud layer and the edge layer is blocked.

4.2.2.3 Cloud Level

According to the limitation of transmission delay caused by physical distance, the cloud level in Iov often pays more attention to the more macro-level computing tasks in the system (such as traffic management, traffic signal control, etc.), rather than focusing on a specific vehicle's computing task. This model can not only ensure that vehicles can respond to emergencies promptly but also ensure that vehicles will not cause system crashes or even traffic accidents due to a lack of calculation data in the case of forced communication interruption.

4.3 Intelligent Manufacturing

Two most important principles in the intelligent manufacturing era are automation and data analysis, the former one of which is the main target and the latter one is one of the most useful tools [16]. In order to follow these principles, intelligent manufacturing should first address response latency, risk control, and privacy protection, and hence requires DL and edge computing. In intelligent factories, edge computing is conducive to expand the computation resources, the network bandwidth, and the storage capacity of the cloud to the IoT edge, as well as realizing the resource scheduling and data processing during manufacturing and production

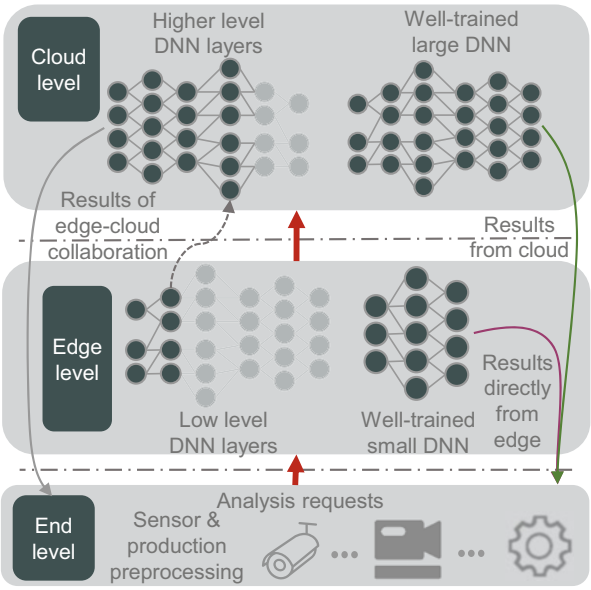

Fig. 4.3 The End–Edge–Cloud collaboration for intelligent manufacturing by AI

[17]. For autonomous manufacturing inspection, *DeepIns* [16] uses DL and edge computing to guarantee performance and process delay, respectively. The main idea of this system is partitioning the DL model, used for inspection, and deploying them on the end, edge, and cloud layer separately for improving the inspection efficiency.

Nonetheless, with the exponential growth of IoT edge devices, (1) how to remotely manage evolving DL models and (2) how to continuously evaluate these models for them are necessary. In [18], a framework, dealing with these challenges, is developed to support complex-event learning during intelligent manufacturing, thus facilitating the development of real-time application on IoT edge devices. Besides, the power, energy efficiency, memory footprint limitation of IoT edge devices [19] should also be considered. Therefore, caching, communication with heterogeneous IoT devices, and computation offloading can be integrated [20] to break the resource bottleneck. In this section, the architecture of Intelligent Manufacturing for edge computing can be showed in Fig. 4.3.

4.3.1 Machine Learning Solution

Machine learning is an important technology of artificial intelligence and has great value in the field of intelligent manufacturing. As we all know, machine learning refers to the ability of a computer to understand and learn inside a physical system through data-based computing algorithms. Therefore, in terms of manufacturing systems, the implementation of machine learning algorithms makes

it possible for machines or other equipment to automatically learn their baselines and working conditions, and create and upgrade the knowledge base throughout the manufacturing process. There are some examples of ML methods which includes data mining, statistical pattern recognition algorithms, and artificial neural network (ANN) to make value in intelligent manufacturing [21].

4.3.2 Deep Learning Solution

The deep integration of the new generation of AI technology which is represented by deep learning, and advanced manufacturing technology has led to the formation of a new generation of intelligent manufacturing. The new generation of intelligent manufacturing that combines the technology and ideas of deep learning will reshape all processes of the entire product cycle, including design, manufacturing, and service and the integration of these processes. It will promote the emergence of new products, new technologies, new models, and new business forms, and it will profoundly affect and change human production methods, production structures, thinking patterns, and lifestyles, and will ultimately greatly increase social productivity[22]. In the future development of intelligent manufacturing, deep learning will bring revolutionary changes to the manufacturing industry and will become one of the important driving forces for the future development of the industry.

4.3.2.1 End Level

For an application scenario such as intelligent manufacturing, we need to ensure that all products in the production line do not have defects, and monitor the operating status of each production device in the production line in real time. Therefore, we divide end devices in this scenario into two groups: one group is mainly composed of cameras and sensors, which aim to make the quality inspection of each product; the other group is mainly various production equipment in the production line, which will upload its device information in an orderly manner at a uniform time interval.

4.3.2.2 Edge Level

Due to the particularity of the intelligent manufacturing scenario, we are more inclined to pay attention to the ability of the entire system to control danger and respond in time, which also requires that devices at the edge level should predict or detect anomalies in the entire system as early as possible. We prefer to allow the system senses and processes the abnormal state in advance, which indicates that the edge level needs to predict the future state trend of production equipment in a

short time relying on DL. It is the fact that the more accurate the prediction result, the longer the effective period of the prediction, and the higher the risk control capability of the entire system, which is the key that the edge level needs to pay attention at present.

4.3.2.3 Cloud Level

We prefer to implement model training and centralized monitoring of equipment in the cloud, because of its rich computing power and high transmission delay between the edge and cloud. Cloud continuously receives data provided by the edge level to train new matching templates and sends it to the edge level for future model prediction, which can not only ensure the accuracy of the entire system's prediction due to the powerful computing resources in the cloud but also ensure the timeliness of its prediction due to the lower latency of the edge level, thereby more fully satisfying the intelligence Technical requirements for manufacturing.

4.4 Smart Home and City

The popularity of IoTs will bring more and more intelligent applications to home life, such as intelligent lighting control systems, smart televisions, and smart air conditioners. But at the same time, smart homes need to deploy numerous wireless IoT sensors and controllers in corners, floors, and walls. For the protection of sensitive home data, the data processing of smart home systems must rely on edge computing. Like use cases in [23, 24], edge computing is deployed to optimize indoor positioning systems and home intrusion monitoring so that they can get lower latency than using cloud computing as well as the better accuracy. Further, the combination of DL and edge computing can make these intelligent services become more various and powerful. For instance, it endows robots the ability of dynamic visual servicing [25] and enables efficient music cognition system [26].

If the smart home is enlarged to a community or city, public safety, health data, public facilities, transportation, and other fields can benefit. The original intention of applying edge computing in smart cities is more due to cost and efficiency considerations. The natural characteristic of geographically distributed data sources in cities requires an edge computing-based paradigm to offer location-awareness and latency-sensitive monitoring and intelligent control. For instance, the hierarchical distributed edge computing architecture in [27] can support the integration of massive infrastructure components and services in future smart cities. This architecture can not only support latency-sensitive applications on end devices but also perform slightly latency-tolerant tasks efficiently on edge nodes, while large-scale DL models responsible for deep analysis are hosted on the cloud. Besides, DL can be utilized to orchestrate and schedule infrastructures to achieve

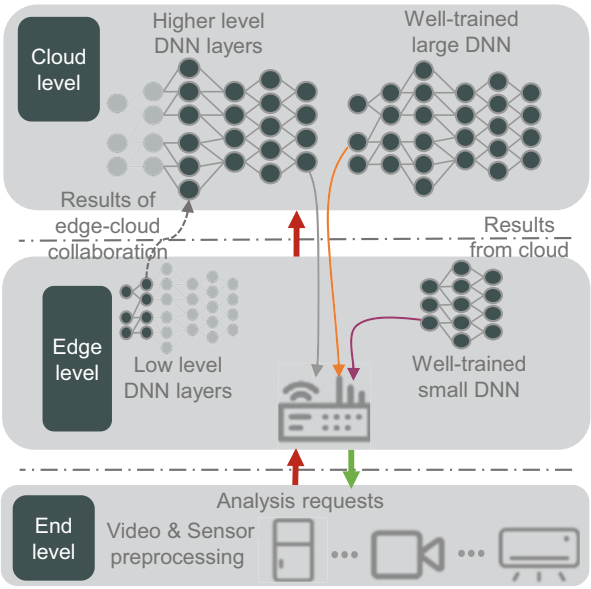

Fig. 4.4 The End–Edge–Cloud collaboration for smart home and city by AI

the holistic load balancing and optimal resource utilization among a region of a city (e.g., within a campus [28]) or the whole city. In this section, the architecture of Edge AI for smart home and city can be presented as Fig. 4.4.

4.4.1 Machine Learning Solution

The large quantities of data are generated at unprecedented rates because of the development of smart home and city and their fast-paced deployment. However, people waste most of the generated data before they extract potentially useful information and knowledge from the data because they have not established mechanisms and standards that benefit from the availability of such data. Therefore, we can use the new generation of machine learning methods to meet the highly dynamic requirements of smart home and city. These methods should be flexible and adaptable to the dynamics of data to perform analysis and learn from real-time data. For example, we can adapt several shallow machine learning approaches, including unsupervised and semi-supervised methods (K-nearest neighbors, support vector machines, etc.) in the level of IoT infrastructure to satisfy the resource limitation of these devices [29].

4.4.2 Deep Learning Solution

The solution of deep learning is often used in the scenarios that need to extract high-level abstractions from the raw data. During the construction of the smart home and city, there have been many application scenarios that need to extract the depth features of data. For example, we can integrate large-scale machine learning which is mainly deep learning, data mining frameworks with semantic learning and ontologies to extract advanced insights and patterns from the generated data in the cloud computing level of smart city architecture. Recent advancements in graphics processing unit (GPU) technology as well as the development of efficient neural network parameter initialization algorithms (e.g., auto-encoders) help to realize efficient deeper learning models and improve the work in the field of smart home and city [29].

4.4.2.1 End Level

When our application scenario is a smart home, it is mainly data acquisition devices for our end devices such as cameras and sensors. Meanwhile, there are more types of end devices in a smart home environment than in a single IoV scenario. It means high requirements on the data structure that the end level provides to the edge level. At the same time, because the dimensions of the data uploaded by the end level are various, we need to consider different deep learning models at the edge level. On the other hand, if our application scenario is extended to smart cities, the types and number of devices included in the end level will increase exponentially. This further requires the edge level's ability to bear computation tasks.

4.4.2.2 Edge Level

In this scenario, users pay more attention to the data privacy and security of the system, which means that most of the computing tasks should be completed at the edge, to minimize the transmission of data in the network. It hopes that the edge only uploads abnormal results or update results to the cloud level, and the cloud level is only responsible for a few of data processing tasks. Corresponding to people wanting to introduce artificial intelligence technology into every aspect of daily life, the edge will use various end devices to achieve the linkage of IoT devices in smart homes or smart cities. For example, when a temperature or smoke sensor in a home detects an abnormality, the edge gateway will start a camera to determine if there is a suspicious fire in the home, edge gateway reports the emergency to the homeowner and management center for avoiding disaster worse.

4.4.2.3 Cloud Level

As the cloud level participates in the calculation of the user's privacy data as little as possible, it means that the edge needs to undertake most of the computing resources required for deep learning. To ensure the efficient use of computing power resources of the edge, it is a good choice to assign the early training tasks of the deep learning model to the cloud server to complete. On the other hand, when the application scenario is constantly expanding, with the increasing number of end devices, the cloud needs to consider the coordination problem of all devices, and also need to monitor the data information uploaded by the edge level in all sub-scenarios.

References

1. P. Viola, M. Jones, Rapid object detection using a boosted cascade of simple features, in *Proceeding of International Conference on Computer Vision and Pattern Recognition*, vol. 1 (2001), pp. 511–518
2. B. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in *Proceedings of Imaging Understanding Workshop* (1981), pp. 121–130
3. J. Ren, Y. Guo, D. Zhang, et al., Distributed and efficient object detection in edge computing: challenges and solutions. IEEE Netw. **32**(6), 137–143 (2018)
4. C. Liu, Y. Cao, Y. Luo, et al., A new deep learning-based food recognition system for dietary assessment on an edge computing service infrastructure. IEEE Trans. Serv. Comput. **11**(2), 249–261 (2018)
5. D. Li, T. Salonidis, N.V. Desai, M.C. Chuah, DeepCham: collaborative edge-mediated adaptive deep learning for mobile object recognition, in *Proceeding of the First ACM/IEEE Symposium on Edge Computing (SEC 2016)* (2016), pp. 64–76
6. B. Fang, X. Zeng, M. Zhang, NestDNN: resource-aware multi-tenant on-device deep learning for continuous mobile vision, in *Proceeding of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom 2018)* (2018), pp. 115–127
7. S. Yi, Z. Hao, Q. Zhang, et al., LAVEA: Latency-aware video analytics on edge computing platform, in *Proceeding of the Second ACM/IEEE Symposium on Edge Computing (SEC 2017)* (2017), pp. 1–13
8. S.Y. Nikouei, Y. Chen, S. Song, et al., Smart Surveillance as an Edge Network Service: From Harr-Cascade, SVM to a Lightweight CNN, in *IEEE 4th International Conference on Collaboration and Internet Computing (CIC 2018)* (2018), pp. 256–265
9. P. Liu, B. Qi, S. Banerjee, EdgeEye - an edge service framework for real-time intelligent video analytics, in *Proceeding of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2018)* (2018), pp. 1–6
10. C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, M. Philipose, VideoEdge: processing camera streams using hierarchical clusters, in *Proceeding of 2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)* (2018), pp. 115–131
11. Y. He, N. Zhao, et al., Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **67**(1), 44–55 (2018)
12. Q. Qi, Z. Ma, Vehicular Edge Computing via Deep Reinforcement Learning (2018). arXiv preprint:1901.04290
13. L.T. Tan, R.Q. Hu, Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning. IEEE Trans. Veh. Technol. **67**(11), 10,190–10,203 (2018)
14. M. Sarwar, M. Christopher, et al., *Machine Learning at the Network Edge: A Survey* (2019)

15. S. Liu, L. Liu, et al., Edge computing for autonomous driving: opportunities and challenges. Proc. IEEE **107**(8), 1697–1716 (2019)
16. L. Li, K. Ota, M. Dong, Deep learning for smart industry: efficient manufacture inspection system with fog computing. IEEE Trans. Ind. Inf. **14**(10), 4665–4673 (2018)
17. L. Hu, Y. Miao, G. Wu, et al., iRobot-Factory: an intelligent robot factory based on cognitive manufacturing and edge computing. Future Gener. Comput. Syst. **90**, 569–577 (2019)
18. J.A.C. Soto, M. Jentsch, et al., CEML: mixing and moving complex event processing and machine learning to the edge of the network for IoT applications, in *Proceeding of the 6th International Conference on the Internet of Things (IoT 2016)* (2016), pp. 103–110
19. G. Plastiras, M. Terzi, C. Kyrkou, T. Theocharidcs, Edge intelligence: challenges and opportunities of near-sensor machine learning applications, in *Proc. IEEE 29th International Conference on Application-specific Systems, Architectures and Processors (ASAP 2018)* (2018), pp. 1–7
20. Y. Hao, Y. Miao, Y. Tian, et al., Smart-Edge-CoCaCo: AI-Enabled Smart Edge with Joint Computation, Caching, and Communication in Heterogeneous IoT (2019). arXiv preprint:1901.02126
21. Y. Chen, Integrated and intelligent manufacturing: perspectives and enablers. Engineering **3**(5), 588–595 (2017)
22. J. Zhou, P. Li, Y. Zhou, B. Wang, J. Zang, L. Meng, Toward new-generation intelligent manufacturing. Engineering **4**(1), 11–20 (2018)
23. S. Liu, P. Si, M. Xu. et al., Edge big data-enabled low-cost indoor localization based on Bayesian analysis of RSS, in *Proceeding of 2017 IEEE Wireless Communications and Networking Conference (WCNC 2017)* (2017), pp. 1–6
24. A. Dhakal, et al., Machine learning at the network edge for automated home intrusion monitoring, in *Proceeding of IEEE 25th International Conference on Network Protocols (ICNP 2017)* (2017), pp. 1–6
25. N. Tian, J. Chen, M. Ma, et al., A Fog Robotic System for Dynamic Visual Servoing (2018). arXiv preprint:1809.06716
26. L. Lu, L. Xu, B. Xu, et al., Fog computing approach for music cognition system based on machine learning algorithm. IEEE Trans. Comput. Social Syst. **5**(4), pp. 1142–1151 (2018)
27. B. Tang, Z. Chen, G. Hefferman, et al.. Incorporating intelligence in Fog computing for big data analysis in smart cities. IEEE Trans. Ind. Inf. **13**(5), 2140–2150 (2017)
28. Y.-C. Chang, Y.-H. Lai, Campus edge computing network based on IoT street lighting nodes. IEEE Syst. J. **14**(1), 164–171 (2020)
29. M. Mohammadi, A. Al-Fuqaha, Enabling cognitive smart cities using big data and machine learning: approaches and challenges. IEEE Commun. Mag. **56**(2), 94–101 (2018)

Chapter 5
Artificial Intelligence Inference in Edge

Abstract In order to further improve the accuracy, DNNs become deeper and require larger-scale dataset. By this means, dramatic computation costs are introduced. Certainly, the outstanding performance of AI is inseparable from the support of high-level hardware, and it is difficult to deploy them in the edge with limited resources. Therefore, large-scale AI models are generally deployed in the cloud while end devices just send input data to the cloud and then wait for the AI inference results. However, the cloud-only inference limits the ubiquitous deployment of AI services. Specifically, it cannot guarantee the delay requirement of real-time services, e.g., real-time detection with strict latency demands. Moreover, for important data sources, data safety and privacy protection should be addressed. To deal with these issues, AI services tend to resort to edge computing. Therefore, AI models should be further customized to fit in the resource-constrained edge, while carefully treating the trade-off between the inference accuracy and the execution latency of them.

5.1 Optimization of AI Models in Edge

AI tasks are usually computationally intensive and require large memory footprints. But in the edge, there are not enough resources to support raw large-scale AI models. Optimizing AI models and quantizing their weights can reduce resource costs. In fact, model redundancies are common in DNNs [1, 2] and can be utilized to make model optimization possible. The most important challenge is how to ensure that there is no significant loss in model accuracy after being optimized. In other words, the optimization approach should transform or re-design AI models and make them fit in edge devices, with as little loss of model performance as possible. In this section, optimization methods for different scenarios are discussed: (1) general optimization methods for edge nodes with relatively sufficient resources; (2) fine-grained optimization methods for end devices with tight resource budgets.

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_5

5.1.1 General Methods for Model Optimization

Increasing the depth and width of AI models with nearly constant computation overhead is one direction of optimization, such as inception [3] and deep residual networks [4] for CNNs. The increase in depth and width of AI models is a direct and effective optimization method. However, the disadvantages of this method are also obvious. Increasing the depth and width of AI models will not only improve performance but also make AI models larger and more complex. So this optimization method will make AI models more difficult to train, which may lead to the consumption of more hardware resources and cause additional training delays.

To solve this problem, the researchers have made a lot of efforts. In [5], the existing methods are divided into four categories:

- *Parameter Pruning and Sharing*: The huge number of parameters is an important factor restricting the training efficiency of AI models. Therefore, in order to achieve more efficient and fast training of AI models, some researchers have optimized the AI models by parameter pruning and sharing. S. Han's team get "free lunch" by keeping only the important connections [6]. In [7], the authors cache the intermediate data between adjacent layers to minimize data movement. In addition, quantization like binarization is a good branch. In 2015, BinaryConnect [8] is introduced, which exploits binary weights in expectation of getting rid of the multiplication and 32x memory saving (if 32 bits single-float precision before). Then, binarized neural networks (BNNS) were developed. In XNOR-Net [9], not only are the filters approximated with binary values, but the input of convolutional layers is binary. And convolutions are approximated primarily by binary operations. These measures provide speedup by a factor of 58 while realizing the similar accuracy on some datasets like CIFAR-10. To make BNNs perform well in small embedded devices, embedded binarized neural networks (eBNNs) were proposed in [10]. The authors try to minimize the memory used for temporaries and reorder the operations in inference. They achieve efficient inference within tens of ms on embedded system with tens of kb memory. It can meet the basic requirements of industrial practice in the real world.
- *Low-rank Factorization*: Since the data usually contains a lot of redundant information, this will be detrimental to the training of AI models. Low-rank factorization is a method to optimize AI models by removing redundant information and it usually includes three methods: SVD decomposition, Tucker decomposition, and block term decomposition. Some researchers have also implemented this. Low-rank approximations with singular value decomposition (SVD) is a main way used in [1].
- *Transferred/Compact Convolution Filters*: In order to achieve the optimization of AI models, parameters can be saved by designing a convolution filter with a special structure. But this method is only suitable for convolutional layers. In [11], SqueezeNet with 50x fewer parameters is 510x smaller than AlexNet and achieves the same level of accuracy on ImageNet by three strategies. Reducing

the parameter is by using 1×1 filters instead of 3×3 filters and decreasing the input channels' number. At the same time, downsampling late in the network is to maximize the accuracy. In addition, depth-wise separable convolutions is a good way to build a lightweight deep neural network used in MobileNets [12] and smart surveillance [13].

- *Knowledge Distillation*: In [14], the concept of knowledge distillation is first proposed and it is a method of transferring knowledge from complex AI models to compact AI models. In general, complex AI models are powerful, while compact AI models are more flexible and efficient. Knowledge distillation can use a complex AI model to train a compact AI model and make it have similar performance to the complex AI models.

These approaches can be applied to different kinds of DNNs or be composed to optimize a complex AI model for the edge.

5.1.2 Model Optimization for Edge Devices

There are many different limitations and requirements for running AI computation tasks on various edge devices. In addition to limited computing and memory footprint, other factors such as network bandwidth and power consumption also need to be considered. Obviously, AI models should be rationally modified and optimized, according to the hardware and software characteristics of a specific edge device, in order to be deployed on it. In this section, efforts for running AI on edge devices are differentiated and discussed.

- *Model Input*: Each application scenario has specific optimization spaces. Concerning object detection, *FFS-VA* uses two prepositive stream-specialized filters and a small full-function tiny-YOLO model to filter out vast but non-target-object frames [15]. In order to adjust the configuration of the input video stream (such as frame resolution and sampling rate) online with low cost, *Chameleon* [16] greatly saves the cost of searching the best model configuration by leveraging temporal and spatial correlations of the video inputs, and allows the cost to be amortized over time and across multiple video feeds. Besides, as depicted in Fig. 5.1,

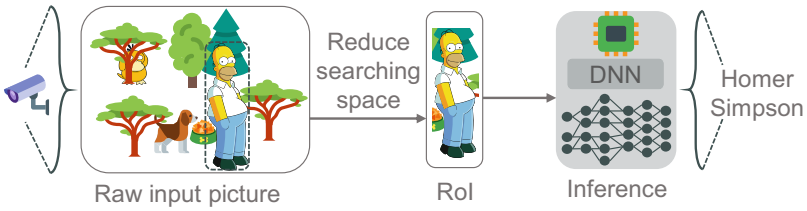

Fig. 5.1 Optimization for model inputs, e.g., narrowing down the searching space of AI models (pictures are with permission from [17])

narrowing down the classifier's searching space [18] and dynamic Region-of-Interest (RoI) encoding [19] to focus on target objects in video frames can further reduce the bandwidth consumption and data transmission delay. Though this kind of methods can significantly compress the size of model inputs and hence reduce the computation overhead without altering the structure of AI models, it requires a deep understanding of the related application scenario to dig out the potential optimization space.

- *Model Structure*: Not paying attention to specific applications, but focusing on the widely used DNNs' structures is also feasible. For instance, point-wise group convolution and channel shuffle [20], paralleled convolution and pooling computation [21], depth-wise separable convolution [13] can greatly reduce computation cost while maintaining accuracy. *NoScope* [22] leverages two types of models rather than the standard model (such as YOLO [23]): specialized models that waive the generality of standard models in exchange for faster inference, and difference detectors that identify temporal differences across input data. After performing efficient cost-based optimization of the model architecture and thresholds for each model, *NoScope* can maximize the throughput of AI services and by cascading these models. Besides, as depicted in Fig. 5.2, parameters pruning can be applied adaptively in model structure optimization as well [24–26]. Furthermore, the optimization can be more efficient if across the boundary between algorithm, software, and hardware. Specifically, general hardware is not ready for the irregular computation pattern introduced by model optimization. Therefore, hardware architectures should be designed to work directly for optimized models [24].
- *Model Selection*: With various AI models, choosing the best one from available AI models in the edge requires weighing both precision and inference time. In [27], the authors use *k*NN to automatically construct a predictor, composed of AI models arranged in sequence. Then, the model selection can be determined by that predictor along with a set of automatically tuned features of the model input. Besides, combining different compression techniques (such as model pruning), multiple compressed AI models with different trade-offs between

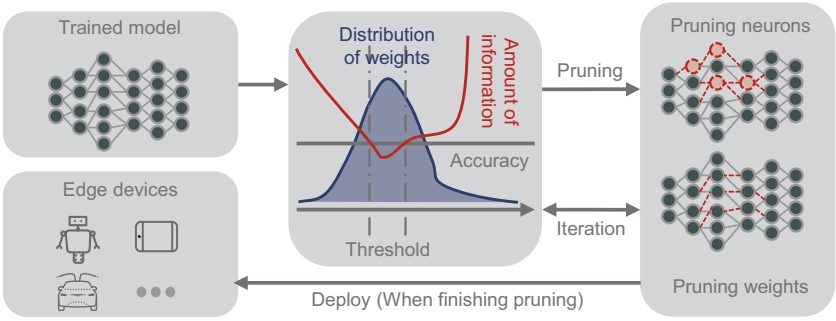

Fig. 5.2 Adaptive parameters pruning in model structure optimization

the performance and the resource requirement can be derived. *AdaDeep* [28] explores the desirable balance between performance and resource constraints, and based on DRL, automatically selects various compression techniques (such as model pruning) to form a compressed model according to current available resources, thus fully utilizing the advantages of them.

- *Model Framework*: Given the high memory footprint and computational demands of DL, running them on edge devices requires expert-tailored software and hardware frameworks. A software framework is valuable if it (1) provides a library of optimized software kernels to enable deployment of DL [29]; (2) automatically compresses AI models into smaller dense matrices by finding the minimum number of non-redundant hidden elements [30]; (3) performs quantization and coding on all commonly used DL structures [25, 30, 31]; (4) specializes AI models to contexts and shares resources across multiple simultaneously executing AI models [31]. With respect to the hardware, running AI models on Static Random Access Memory (SRAM) achieves better energy savings compared to Dynamic RAM (DRAM) [25]. Hence, DL performance can be benefited if underlying hardware directly supports running optimized AI models [32] on the on-chip SRAM.

5.2 Segmentation of AI Models

Artificial intelligence technologies are widely applied in our lives now, which greatly facilitates our lives. However, most of intelligent applications only perform in the cloud and the edge devices just play a role of collecting and uploading the data. This brings a heavy burden to the cloud. It also takes up a lot of network bandwidth especially for video processing applications. Nowadays, with the advancement of technology, the edge devices have better hardware configurations. At the same time, common deep learning model is composed of multilayer neural networks. And there are significant differences in computing resource requirements and the size of output data for different network layers. The researchers begin to think about whether pushing part or all of the computing tasks in the edge by the segmentation of deep learning model. In this way, a large number of computing tasks can be decomposed to different parts. Different device can solve the problem collaboratively.

In [33], the delay and power consumption of the most advanced AI models are evaluated on the cloud and edge devices, finding that uploading data to the cloud is the bottleneck of current AI servicing methods (leading to a large overhead of transmitting). Dividing the AI model and performing distributed computation can achieve better end-to-end delay performance and energy efficiency. In addition, by pushing part of DL tasks from the cloud to the edge, the throughput of the cloud can be improved. Therefore, the AI model can be segmented into multiple partitions and then allocated to (1) heterogeneous local processors (e.g., GPUs, CPUs) on the

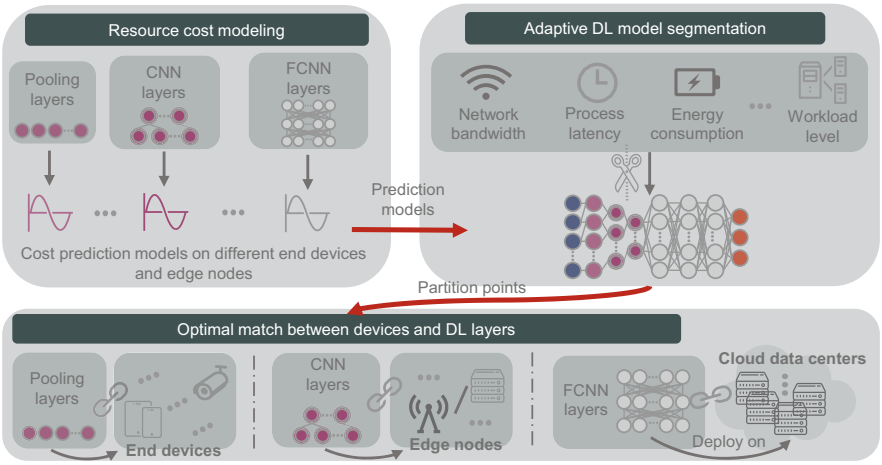

Fig. 5.3 Segmentation of AI models in the edge

end device [34], (2) distributed edge nodes [35, 36], or (3) collaborative "end–edge–cloud" architecture [33, 37–39].

Partitioning the AI model horizontally, i.e., along the end, edge, and cloud, is the most common segmentation method. The process of data analysis is usually divided into two parts [38, 39]. One part is processed in the edge and the other part is processed in the cloud. Because of uploading reduced intermediate data instead of input data [38], this way not only reduces the network traffic between the edge and the cloud but also avoids the risk of security and privacy leakage in data transmission. The challenge lies in how to intelligently select the partition points. As illustrated in Fig. 5.3, a general process for determining the partition point can be divided into three steps [33, 37]: (1) measuring and modeling the resource cost of different DNN layers and the size of intermediate data between layers; (2) predicting the total cost by specific layer configurations and network bandwidth; (3) choosing the best one from candidate partition points according to delay, energy requirements, etc.

Another kind of model segmentation is vertically partitioning particularly for CNNs. In contrast to horizontal partition, vertical partition fuses layers and partitions them vertically in a grid fashion, and thus divides CNN layers into independently distributable computation tasks. DeepThings [36] exploit a new way called Fused Tile Partitioning (FTP). Fused layers are partitioned vertically in a grid fashion. The experiment result displays that FTP can reduce the memory footprint to 32% at least without reducing accuracy. At the same time, the authors implement the dynamic workload allocation on the edge clusters. Similarly, J. Zhang's team design a framework for the locally distributed mobile computing in [35]. They present a universal segmentation tool for neural network layers. And the migration on the working nodes is through the Wireless Local Area Network (WLAN). They test some common neural networks. The experiment of GoogLeNet gets the best

performance. The system almost reduced total latency by half. In DeepX [34], Deep Architecture Decomposition (DAD) is designed to decompose the deep models with large number of units into different types of unit-blocks. Then these blocks are allocated to execute efficiently on local and remote processors.

5.3 Early Exit of Inference (EEoI)

Though model compression and model segmentation can facilitate deploying AI in the edge computing network, however, both of them have disadvantages. The former one may jeopardize the model accuracy irreversibly, while the latter may cause large communication overhead between segmented models. It is challenging to achieve fast and effective inference.

To reach the best trade-off between model accuracy and processing delay, multiple AI models with different model performance and resource cost can be maintained for each AI service. Then, by intelligently selecting the best model, the desired adaptive inference is achieved [40]. Nonetheless, this idea can be further improved by the emerged EEoI. In 2016, BranchyNet [41] achieved the fast inference by withdrawing from a branch of the deep neural network in advance. The researchers found that most of the test samples are able to get enough features in an early layer of the network. Therefore, they added several side branches to the original main branch. When reaching certain conditions, they can exit the inference in the earlier stage from one of the branches, instead of traversing the entire layers of the network. There is no doubt that this method can significantly reduce the computation of inference. BranchyNet can choose the branch with shortest time under a given accuracy requirement. In addition, BranchyNet achieves the regularization via joint optimization of all exit points to prevent overfitting and improve the accuracy, and it mitigates the vanishing gradients. One year later, the same team deploy BranchyNet to distributed deep neural networks (DDNNs) over distributed computing hierarchies in [42]. They exploit shallow portions of the network to implement localized inference via the early exit and the goal is to achieve the fast response in the edge and accurate inference in the cloud. In [43], Edgent resizes the DNN through BranchyNet to accelerate the inference. The result of experiment shows that the optimal exit point is postponed and the accuracy can be improved when the latency requirement is reduced. With the help of adaptive partition, Edgent achieves the collaborative and on-demand DNN co-inference.

It is well known that DNNs are able to extract increasingly better features at each network layer. However, the performance improvement of additional layers in DNNs is at the expense of increased latency and energy consumption in feedforward inference. As DNNs grow larger and deeper, these costs become more prohibitive for edge devices to run real-time and energy-sensitive DL applications. By additional side branch classifiers, for partial samples, EEoI allows inference to exit early via these branches if with high confidence. For more difficult samples, EEoI will use more or all DNN layers to provide the best predictions.

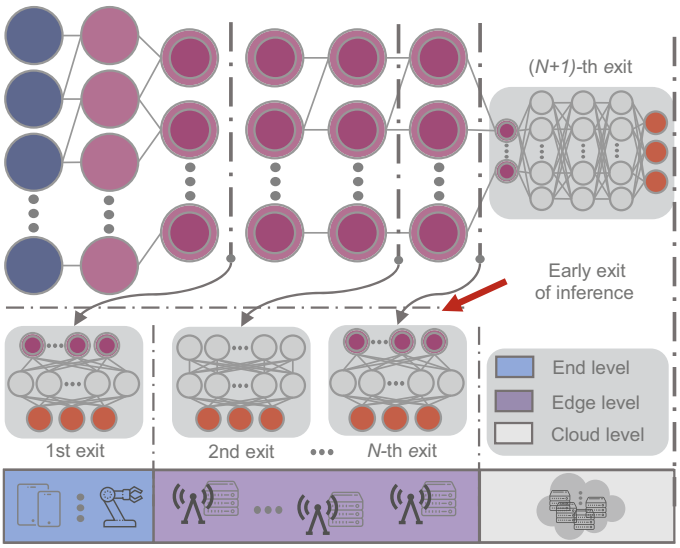

Fig. 5.4 Early exit of inference for DL inference in the edge

As depicted in Fig. 5.4, by taking advantage of EEoI, fast and localized inference using shallow portions of AI models at edge devices can be enabled. By this means, the shallow model on the edge device can quickly perform initial feature extraction and, if confident, can directly give inference results. Otherwise, the additional large AI model deployed in the cloud performs further processing and final inference. Compared to directly offloading DL computation to the cloud, this approach has lower communication costs and can achieve higher inference accuracy than those of the pruned or quantized AI models on edge devices [42, 44]. In addition, since only immediate features rather than the original data are sent to the cloud, it provides better privacy protection. Nevertheless, EEoI shall not be deemed independent to model optimization (Sect. 5.1.2) and segmentation (Sect. 5.2). The envision of distributed DL over the end, edge, and cloud should take their collaboration into consideration, e.g., developing a collaborative and on-demand co-inference framework [43] for adaptive DNN partitioning and EEoI.

5.4 Sharing of AI Computation

AI computation is often complicated and the intensive calculation of it is a great test for device resources. However, AI computation is highly logical which makes the different DL operation processes have a certain correlation. So how to use the correlation of DL operations has become a starting point for optimizing AI models. For the sharing of AI computation, one idea is to cache and reuse the inference

results to avoid redundant operations and this idea has achieved good practical results in some scenarios.

The requests from nearby users within the coverage of an edge node may exhibit spatiotemporal locality [45]. For instance, users within the same area might request recognition tasks for the same object of interest, and it may introduce redundant computation of DL inference. In this case, based on offline analysis of applications and online estimates of network conditions, *Cachier* [45] proposes to cache related AI models for recognition applications in the edge node and to minimize expected end-to-end latency by dynamically adjusting its cache size. Therefore, when the AI model in the cache can meet the requirements of the request, the AI model can be directly obtained from the cache for use. In this way, redundant operations can be avoided by using caching and reuse.

In addition, not only can the AI model be cached directly, a more fine-grained cache is also very effective. For example, during the calculation process inside the model, some calculation results can be cached to reduce the calculation amount. Based on the similarity between consecutive frames in first-person-view videos, *DeepMon* [46] and *DeepCache* [47] utilize the internal processing structure of CNN layers to reuse the intermediate results of the previous frame to calculate the current frame, i.e., caching internally processed data within CNN layers, to reduce the processing latency of continuous vision applications.

Nevertheless, to proceed with effective caching and results reusing, accurate lookup for reusable results shall be addressed, i.e., the cache framework must systematically tolerate the variations and evaluate key similarities. *DeepCache* [47] performs cache key lookup to solve this. Specifically, it divides each video frame into fine-grained regions and searches for similar regions from cached frames in a specific pattern of video motion heuristics. For the same challenge, *FoggyCache* [48] first embeds heterogeneous raw input data into feature vectors with generic representation. Then, Adaptive Locality Sensitive Hashing (A-LSH), a variant of LSH commonly used for indexing high-dimensional data, is proposed to index these vectors for fast and accurate lookup. At last, Homogenized kNN, which utilizes the cached values to remove outliers and ensure a dominant cluster among the k records initially chosen, is implemented based on kNN to determine the reuse output from records looked up by A-LSH. Therefore, by accurate lookup for reusable results and caching the calculation results, the calculation amount of the AI model can be reduced and the pressure on hardware resources can be alleviated. However, the method of sharing of is not only the cache of calculation results, it also includes other thinking directions.

Differ from sharing inference results, *Mainstream* [49] proposes to adaptively orchestrate DNN stem-sharing (the common part of several specialized AI models) among concurrent video processing applications. By exploiting computation sharing of specialized models among applications trained through TL from a common DNN stem, aggregate per-frame compute time can be significantly decreased. Though more specialized AI models mean both higher model accuracy and less shared DNN stems, the model accuracy decreases slowly as less-specialized AI models are employed (unless the fraction of the model specialized is very small). This

characteristic hence enables large portions of the AI model can be shared with low accuracy loss in *Mainstream*. Similarly, in [50], the author reduced the computation by sharing between different AI models. By considering the correlation between training samples, the author proposes a transfer learning algorithm in the same target region, that is, if there are multiple related AI models in a target region, then training of one AI model can also benefit other related AI models. This method can reduce the AI computation of untrained AI models in the same target region by sharing from a well-trained AI model.

References

1. E. Denton, et al., Exploiting linear structure within convolutional networks for efficient evaluation, in *Advances in Neural Information Processing Systems 27 (NeurIPS 2014)* (2014), pp. 1269–1277
2. W. Chen, J. Wilson, S. Tyree, et al., Compressing neural networks with the Hashing Trick, in *Proceeding of the 32nd International Conference on International Conference on Machine Learning (ICML 2015)* (2015), pp. 2285–2294
3. C. Szegedy, W. Liu, Y. Jia, et al., Going deeper with convolutions, in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015)* (2015), pp. 1–9
4. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)* (2016), pp. 770–778
5. Y. Cheng, D. Wang, P. Zhou, T. Zhang, A Survey of Model Compression and Acceleration for Deep Neural Networks (2017). arXiv preprint:1710.09282
6. S. Han, J. Pool, J. Tran, et al., Learning both weights and connections for efficient neural networks, in *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)* (2015), pp. 1135–1143
7. M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2016)* (2016), pp. 1–12
8. M. Courbariaux, Y. Bengio, J.-P. David, BinaryConnect: training deep neural networks with binary weights during propagations, in *Advances in Neural Information Processing Systems 28 (NeurIPS 2015)* (2015), pp. 3123–3131
9. M. Rastegari, V. Ordonez, et al., XNOR-Net: ImageNet classification using binary convolutional neural networks, in *2018 European Conference on Computer Vision (ECCV 2016)* (2016), pp. 525–542
10. B. Mcdanel, Embedded binarized neural networks, in *Proceeding of the 2017 International Conference on Embedded Wireless Systems and Networks (EWSN 2017)* (2017), pp. 168–173
11. F.N. Iandola, S. Han, M.W. Moskewicz, et al., SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and < 0.5 MB Model Size (2016). arXiv preprint:1602.07360
12. A.G. Howard, M. Zhu, B. Chen, et al., MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications (2017). arXiv preprint:1704.04861
13. S.Y. Nikouei, Y. Chen, S. Song, et al., Smart surveillance as an edge network service: from Harr-Cascade, SVM to a lightweight CNN, in *IEEE 4th International Conference on Collaboration and Internet Computing (CIC 2018)* (2018), pp. 256–265
14. G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network (2015). arXiv preprint:1503.02531
15. C. Zhang, Q. Cao, H. Jiang, et al., FFS-VA: a fast filtering system for large-scale video analytics, in *Proceeding of the 47th International Conference on Parallel Processing (ICPP 2018)* (2018), pp. 1–10

16. J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, I. Stoica, Chameleon: scalable adaptation of video analytics, in *Proceeding of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2018)* (2018), pp. 253–266
17. Fox, Homer simpson. https://simpsons.fandom.com/wiki/File:Homer_Simpson.svg
18. S.Y. Nikouei, et al., Real-time human detection as an edge service enabled by a lightweight CNN, in *2018 IEEE International Conference on Edge Computing (IEEE EDGE 2018)* (2018), pp. 125–129
19. L. Liu, H. Li, M. Gruteser, Edge assisted real-time object detection for mobile augmented reality, in *Proceeding of the 25th Annual International Conference on Mobile Computing and Networking (MobiCom 2019)* (2019), pp. 1–16
20. X. Zhang, X. Zhou, M. Lin, J. Sun, ShuffleNet: an extremely efficient convolutional neural network for mobile devices, in *Proceeding of 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR 2018)* (2018), pp. 6848–6856
21. L. Du, et al., A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things. IEEE Trans. Circuits Syst. I Regul. Pap. **65**(1), 198–208 (2018)
22. D. Kang, J. Emmons, F. Abuzaid, P. Bailis, M. Zaharia, NoScope: optimizing neural network queries over video at scale. Proc. VLDB Endow. **10**(11), 1586–1597 (2017)
23. J. Redmon, S. Divvala, et al., You only look once: unified, real-time object detection, in *Proceeding of 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016)* (2016), pp. 779–788
24. S. Han, Y. Wang, H. Yang, et al., ESE: efficient speech recognition engine with sparse LSTM on FPGA, in *Proceeding of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017)* (2017), pp. 75–84
25. S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural networks with pruning, trained quantization and Huffman Coding, in *Proceeding of the 6th International Conference on Learning Representations (ICLR 2016)* (2016)
26. S. Bhattacharya, N.D. Lane, Sparsification and separation of deep learning layers for constrained resource inference on wearables, in *Proceeding of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM (SenSys 2016)* (2016), pp. 176–189
27. B. Taylor, V.S. Marco, W. Wolff, et al., Adaptive deep learning model selection on embedded systems, in *Proceeding of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2018)* (2018), pp. 31–43
28. S. Liu, Y. Lin, Z. Zhou, et al., On-demand deep model compression for mobile devices, in *Proceeding of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2018)* (2018), pp. 389–400
29. L. Lai, N. Suda, Enabling deep learning at the IoT edge, in *Proceeding of the International Conference on Computer-Aided Design (ICCAD 2018)* (2018), pp. 1–6
30. S. Yao, Y. Zhao, A. Zhang, et al., DeepIoT: compressing deep neural network structures for sensing systems with a compressor-critic framework, in *Proceeding of the 15th ACM Conference on Embedded Network Sensor Systems (SenSys 2017)* (2017), pp. 1–14
31. S. Han, H. Shen, M. Philipose, et al., MCDNN: an execution framework for deep neural networks on resource-constrained devices, in *Proceeding of the 14th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2016)* (2016), pp. 123–136
32. S. Han, et al., EIE: efficient inference engine on compressed deep neural network, in *ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA 2016)* (2016), pp. 243–254
33. Y. Kang, J. Hauswald, C. Gao, et al., Neurosurgeon: collaborative intelligence between the cloud and mobile edge, in *Proceeding of 22nd International Conference Architecture Support Programming Language Operator System (ASPLOS 2017)* (2017), pp. 615–629
34. N.D. Lane, S. Bhattacharya, P. Georgiev, et al., DeepX: a software accelerator for low-power deep learning inference on mobile devices, in *15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2016)* (2016), pp. 1–12

35. J. Zhang, et al., A Locally Distributed Mobile Computing Framework for DNN based Android Applications, in *Proceeding of the Tenth Asia-Pacific Symposium on Internetware (Internetware 2018)* (2018), pp. 1–6
36. Z. Zhao, K.M. Barijough, A. Gerstlauer, DeepThings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2348–2359 (2018)
37. Z. Zhao, Z. Jiang, N. Ling, et al., ECRT: an edge computing system for real-time image-based object tracking, in *Proceeding of the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys 2018)* (2018), pp. 394–395
38. H. Li, K. Ota, M. Dong, Learning IoT in edge: deep learning for the internet of things with edge computing. IEEE Netw. **32**(1), 96–101 (2018)
39. G. Li, L. Liu, X. Wang, et al., Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge, in *Proceeding of International Conference on Artificial Neural Networks (ICANN 2018)* (2018), pp. 402–411
40. S.S. Ogden, T. Guo, MODI: mobile deep inference made efficient by edge computing, in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2018)* (2018)
41. S. Teerapittayanon, et al., BranchyNet: Fast inference via early exiting from deep neural networks, in *Proceeding of the 23rd International Conference on Pattern Recognition (ICPR 2016)* (2016), pp. 2464–2469
42. S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed deep neural networks over the cloud, the edge and end devices, in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)* (2017), pp. 328–339
43. E. Li, Z. Zhou, X. Chen, Edge intelligence: on-demand deep learning model co-inference with device-edge synergy, in *Proceeding of the 2018 Workshop on Mobile Edge Communications (MECOMM 2018)* (2018), pp. 31–36
44. L. Li, K. Ota, M. Dong, Deep learning for smart industry: efficient manufacture inspection system with fog computing. IEEE Trans. Ind. Inf. **14**(10), 4665–4673 (2018)
45. U. Drolia, K. Guo, J. Tan, et al., Cachier: edge-caching for recognition applications, in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)* (2017), pp. 276–286
46. L.N. Huynh, Y. Lee, R.K. Balan, DeepMon: mobile GPU-based deep learning framework for continuous vision applications, in *Proceeding of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2017)* (2017), pp. 82–95
47. M. Xu, M. Zhu, et al., DeepCache: principled cache for mobile deep vision, in *Proceeding of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom 2018)* (2018), pp. 129–144
48. P. Guo, B. Hu, et al., FoggyCache: cross-device approximate computation reuse, in *Proceeding of the 24th Annual International Conference on Mobile Computing and Networking (MobiCom 2018)* (2018), pp. 19–34
49. A.H. Jiang, D.L.-K. Wong, C. Canel, L. Tang, I. Misra, M. Kaminsky, M.A. Kozuch, P. Pillai, D.G. Andersen, G.R. Ganger, Mainstream: dynamic stem-sharing for multi-tenant video processing, in *Proceeding of the 2018 USENIX Conference on USENIX Annual Technical Conference (USENIX ATC 2018)* (2018), pp. 29–41
50. L. Wang, W. Liu, D. Zhang, Y. Wang, E. Wang, Y. Yang, Cell selection with deep reinforcement learning in sparse mobile crowdsensing, in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)* (IEEE, New York, 2018), pp. 1543–1546

Chapter 6
Artificial Intelligence Training at Edge

Abstract Present cloud training (or cloud–edge training) is facing challenges in AI services requiring continuous learning and data privacy. Naturally, the edge architecture, which consists of a large number of edge nodes with modest computing resources, can cater for alleviating the pressure of networks and protecting data privacy by processing the data or training at themselves. Training at the edge or potentially among "end–edge–cloud," treating the edge as the core architecture of training, is called "AI Training at Edge." Such kind of training may require significant resources to digest distributed data and exchange updates in the hierarchical structure. Especially, FL is an emerging distributed learning setting and is promising to address these issues. For devices with diverse capabilities and limited network conditions in edge computing, FL can protect privacy in the time of handling non-IID training data, and has promising scalability in terms of efficient communication, resource optimization and security. As the principal content of this chapter, some selected works on FL are listed in the first table in this chapter.

6.1 Distributed Training at Edge

Present AI training (distributed or not) in the cloud data center, namely cloud training or cloud–edge training [1], is a kind of learning solution that training data are preprocessed at the edge and then transmitted to cloud, which is not appropriate for all kind of AI services, especially when considering continuous learning, geographical dispersed location, and privacy-sensitive data. To start with the discussion of distributed training at edge, we begin with three key words related to AI and edge computing:

(1) ***Continuous learning[2]:*** As a key to current and future AI, continuous learning emphasizes the standing updating of knowledge about environmental information and action strategies, which is well demonstrated in DRL recently.
(2) ***Geographical dispersed location:*** Since massive data are generated from geographical dispersed location and analyzed to achieve a unified goal, *Who performs the computation process of AI training* and *how to reduce communi-*

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_6

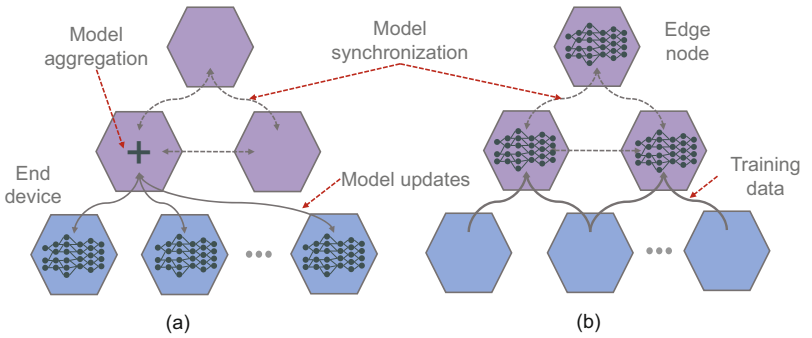

Fig. 6.1 Distributed training at edge environments. (**a**) Distributed training at end devices. (**b**) Distributed training at edge nodes

cation cost among related devices constitute two major concerns in performing AI training at edge.

(3) ***Privacy-sensitive data:*** In addition, present AI training (distributed or not) might violate privacy issues about training data, which is significantly serious in merging all data into the cloud over a large and unsafe network.

For example, with respect to surveillance applications integrated with object detection and target tracking, if end devices directly send a huge amount of real-time monitoring data to the cloud for online training, it will bring about high networking costs and might cause the information leakage about video content. As a solution to these challenges above, distributed training at the edge can be traced back to the work of [3], where a decentralized Stochastic Gradient Descent (SGD) method is proposed for the edge computing network to solve a large linear regression problem. However, this proposed method is designed for seismic imaging application and cannot be generalized for AI training, since the communication cost for training large-scale models is extremely high. In [4], two different distributed learning solutions for edge computing environments are proposed. As depicted in Fig. 6.1, one solution is that each end device trains a model based on local data, and then these model updates are aggregated at edge nodes. Another one is edge nodes train their own local models with the data from end devices, and their model updates are exchanged and refined for constructing a global model. Though large-scale distributed training at edge evades transmitting bulky raw dataset to the cloud, the communication cost for information exchanging between edge devices is inevitably introduced. Besides, in practical, edge devices may suffer from higher latency, lower transmission rate, and intermittent connections, and, therefore, further hindering model updates from different devices. Considering the popularity and abundant works of DL training at edge, we will mainly take DL (and DRL) as examples to introduce their idea and design detail, and we think these learning methods and skills are also valuable for other AI domains (Table 6.1).

Table 6.1 Summary of the selected works on FL

	Ref.	DL model	Scale	Dependency	Main idea	Key metrics or performance
Vanilla FL	[5]	FCNN, CNN, LSTM	Up to 5e5 clients	TensorFlow	Learn a shared model by aggregating training updates instead of datasets	Comm. rounds reduction: 10–100×
	[6]	RNN	Up to 1.5e6 clients	TensorFlow	Pace steering for scalable FL	Scalability improvement: up to 1.5e6 clients
Communication (Comm.)-efficient FL	[7]	ResNet18	7 clusters/4 clients per cluster	–	Gradient sparsification; Periodic averaging	Top 1 accuracy; Comm. latency reduction
	[8]	CNN, LSTM	Up to 1e3 clients	–	Sketched updates	Comm. cost reduction: by two orders of magnitude
	[9]	CNN	Up to 500 clients	TensorFlow	Lossy compression on the global model; Federated Dropout	The reductions in downlink/uplink/local computation: 14×/28×/1.7×
	[10]	CNN, RNN	Up to 37 clients	TensorFlow	Let faster clients continue with their mini-batch training while keeping overall synchronization	Convergence acceleration: 62.4%
	[11]	CNN	5–500 clients (simulation); 5 Devices (testbed)	–	Design a control algorithm that determines the best trade-off between local update and global aggregation	Training accuracy under resource budget
	[12]	FCNN	50 clients	–	Periodic averaging; Partial device participation; Quantized message-passing	Total training loss and time
	[13]	CNN	500 clients	–	Global data distribution based data augmentation; Mediator-based multi-client rescheduling	Top 1 accuracy improvement: 5.59–5.89%; Comm. traffic reduction: 92%

(continued)

Table 6.1 (continued)

	Ref.	DL Model	Scale	Dependency	Main idea	Key metrics or performance
	[14]	LeNet, CNN, VGG11	10 Raspberry Pi	Py(Torch)	Jointly train and prune the model in a federated manner	Comm. and computation load reduction
Resource-optimized FL	[15]	AlexNet, LeNet	Multiple NVIDIA Jetson Nano	–	Partially train the model by masking a particular number of resource-intensive neurons	Training acceleration: 2×; Model accuracy improvement: 4%
	[16]	–	Up to 50 clients	TensorFlow	Jointly optimize FL parameters and resources of user equipment	Convergence rate; Test accuracy
	[17]	–	20 clients/1 BS	–	Jointly optimize wireless resource allocation and client selection	Reduction of the FL loss function value: up to 16%
	[18]	LSTM	23–1101 clients	TensorFlow	Modify FL training objectives with α-fairness	Fairness; Training accuracy
Security-enhanced FL	[19]	CNN	100 clients	MXNet	Use the trimmed mean as a robust aggregation	Top 1 accuracy against data poisoning
	[20]	1-layer Softmax, SqueezeNet, VGGNet	50 clients	–	Use client contribution similarity to alleviate Sybil-based model update poisoning	The expected number of attackers can be arbitrary; Improved training accuracy when under the attack
	[21]	–	Up to 500 nodes	–	Robust designs with two models to alleviate the effects of comm. noise	Prediction accuracy; Loss function value
	[22]	–	2e10–2e14 clients	–	Use secure aggregation to protect the privacy of client's model gradient	Comm. expansion: 1.73×−1.98×
	[23]	–	10 clients	–	Use blockchain to exchange and verify model updates of local training	Learning completion latency

Most of the gradient exchanges are redundant, and hence updated gradients can be compressed to cut down the communication cost while preserving the training accuracy (such as *DGC* in [24]). *First*, *DGC* stipulates that only important gradients are exchanged, i.e., only gradients larger than a heuristically given threshold are transmitted. In order to avoid the information losing, the rest of the gradients are accumulated locally until they exceed the threshold. To be noted, gradients whether being immediately transmitted or accumulated for later exchanging will be coded and compressed, and hence saving the communication cost. *Second*, considering the sparse update of gradients might harm the convergence of DL training, momentum correction and local gradient clipping are adopted to mitigate the potential risk. By momentum correction, the sparse updates can be approximately equivalent to the dense updates. Before adding the current gradient to previous accumulation on each edge device locally, gradient clipping is performed to avoid the exploding gradient problem possibly introduced by gradient accumulation. Certainly, since partial gradients are delayed for updating, it might slow down the convergence. Hence, *finally*, for preventing the stale momentum from jeopardizing the performance of training, the momentum for delayed gradients is stopped, and less aggressive learning rate and gradient sparsity are adopted at the start of training to reduce the number of extreme gradients being delayed.

With the same purpose of reducing the communication cost of synchronizing gradients and parameters during distributed training, two mechanisms can be combined together [25]. The first is transmitting only important gradients by taking advantage of sparse training gradients [26]. Hidden weights are maintained to record times of a gradient coordinate participating in gradient synchronization, and gradient coordinates with large hidden weight value are deemed as important gradients and will be more likely be selected in the next round training. On the other hand, the training convergence will be greatly harmed if residual gradient coordinates (i.e., less important gradients) are directly ignored, hence, in each training round, small gradient values are accumulated. Then, in order to avoid that these outdated gradients only contribute little influence on the training, momentum correction, viz., setting a discount factor to correct residual gradient accumulation, is applied.

Particularly, when training a large DL model, exchanging corresponded model updates may consume more resources. Using an online version of KD can reduce such kind of communication cost [27]. In other words, the model outputs rather the updated model parameters on each device are exchanged, making the training of large-sized local models possible. Besides communication cost, privacy issues should be concerned as well. For example, in [28], personal information can be purposely obtained from training data by making use of the privacy leaking of a trained classifier. The privacy protection of training dataset at the edge is investigated in [29]. Different from [4, 24, 25], in the scenario of [29], training data are trained at edge nodes as well as be uploaded to the cloud for further data analysis. Hence, Laplace noises [30] are added to these possibly exposed training data for enhancing the training data privacy assurance. In detail, security issues raised by data privacy in FL will be discussed in Sect. 6.5.

6.2 Vanilla Federated Learning at Edge

In Sect. 6.1, the holistic network architecture is explicitly separated, specifically, training is limited at the end devices or the edge nodes independently instead of among both of them. Certainly, by this means, it is simple to orchestrate the training process since there is no need to deal with heterogeneous computing capabilities and networking environments between the end and the edge. Nonetheless, DL training should be ubiquitous as well as DL inference. Federated Learning (FL) [5, 6] is emerged as a practical DL training mechanism among the end, the edge, and the cloud. Though in the framework of native FL, modern mobile devices are taken as the clients performing local training. Naturally, these devices can be extended more widely in edge computing [19, 31]. End devices, edge nodes, and servers in the cloud can be equivalently deemed as clients in FL. These clients are assumed capable of handling different levels of DL training tasks, and hence contribute their updates to the global DL model. In this section, fundamentals of FL are discussed.

Without requiring uploading data for central cloud training, FL [5, 6] can allow ubiquitous devices in edge computing to train local DL models with the data they have and upload only the updated model instead. As depicted in Fig. 6.2, there are two roles in vanilla FL: clients with the local data and an aggregation server responsible for model aggregation (use "server" in following). FL iteratively solicits a random set of clients to (1) download the global DL model from the server, (2) train their local models on the downloaded global model with their local data, and (3) upload only the updated model to the server for model averaging. According to the relationship between the two roles in FL and the three levels in edge computing, we list three feasible solutions for FL training at edge: (1) End–edge cooperation: An edge node replaces the cloud as the server. The interaction between "end" and "edge" can benefit from the low latency and high bandwidth of edge computing. (2) Edge–cloud cooperation: Edge nodes obtain training data from end devices (or collect the data themselves) then participate in FL as clients, and the cloud can be thought of as the server. This solution saves valuable computing resources and

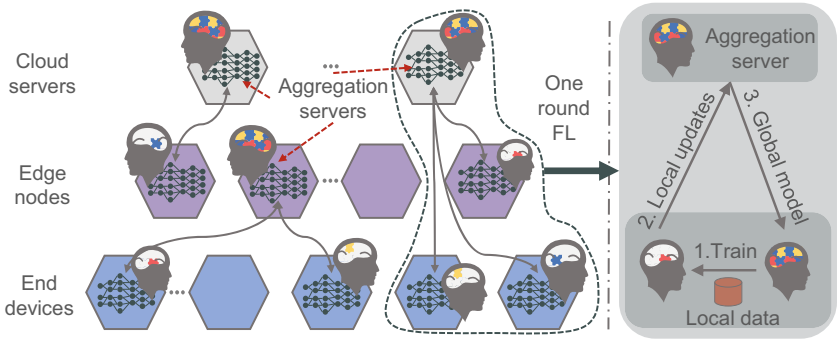

Fig. 6.2 Federated learning among hierarchical network architectures

energy costs of end devices by avoiding the execution of training process on end devices, and limits the sharing scope of the data from end devices. (3) End–edge co-training: Both end devices and edge nodes perform model training as clients, and the cloud performs the aggregation, which can combine the advantages of the above two solutions.

Privacy and security risks can be significantly reduced by restricting training data to corresponding training devices, and thus avoiding the privacy issues as in [28], incurred by uploading training data to the cloud. Besides, FL introduces *FederatedAveraging* to combine local SGD on each device with a server performing model averaging. Experimental results corroborate *FederatedAveraging* is robust to unbalanced and non-IID data and can facilitate the training process, viz., reducing the rounds of communication needed to train a DL model.

To summarize, FL can deal with several key challenges in edge computing networks: (1) **Non-IID training data**. Training data on each device is collected by itself, or received from a small portion of all data sources. Hence, any individual training data of a device will not be able to represent the global one. In FL, this can be met by *FederatedAveraging*; (2) **Limited communication**. Devices might potentially be offline or located in a poor communication environment. Nevertheless, performing more training computation on resource-sufficient devices can cut down communication rounds needed for global model training. In addition, FL only selects a part of devices to upload their updates in one round, therefore successfully handling the circumstance where devices are unpredictably offline; (3) **Unbalanced contribution**. It can be tackled by *FederatedAveraging*, specifically, some devices may have less free resources for FL, resulting in varying amounts of training data and training capability among devices; (4) **Privacy and security**. FL enables training devices to perform distributed DL training by model updating instead of data uploading, which reduces the risk of information leakage and the impact of unexpected events. Further, secure aggregation and differential privacy [30], which are useful in avoiding the disclosure of privacy-sensitive data contained in local updates, can be applied naturally.

6.3 Communication-Efficient FL

In FL, raw training data are not required to be uploaded, thus largely reducing the communication cost. However, FL still needs to transmit locally updated models to the central server. Supposing the DL model size is large enough, uploading updates, such as model weights, from edge devices to the central server may also consume nonnegligible communication resources. To meet this, we can let FL clients communicate with the central server periodically (rather continually) to seek consensus on the shared DL model [7]. In addition, *structured update*, *sketched update* can help enhance the communication efficiency when clients uploading updates to the server as well. *Structured update* means restricting the model updates to have a pre-specified structure, specifically, (1) low-rank matrix; or (2) sparse

matrix [7, 8]. On the other hand, for *sketched update*, full model updates are maintained. But before uploading them for model aggregation, combined operations of subsampling, probabilistic quantization, and structured random rotations are performed to compress the full updates [8]. *FedPAQ* [12] simultaneously incorporates these features and provides near-optimal theoretical guarantees for both strongly convex and non-convex loss functions, while empirically demonstrating the communication–computation trade-off.

Different from only investigating on reducing communication cost on the uplink, [9] takes both server-to-device (downlink) and device-to-server (uplink) communication into consideration. For the downlink, the weights of the global DL model are reshaped into a vector, and then subsampling and quantization are applied [8]. Naturally, such kind of model compression is lossy, and unlike on the uplink (multiple edge devices are uploading their models for averaging), the loss cannot be mitigated by averaging on the downlink. *Kashin's representation* [32] can be utilized before subsampling as a basis transform to mitigate the error incurred by subsequent compression operations. Furthermore, for the uplink, each edge device is not required to train a model based on the whole global model locally, but only to train a smaller sub-model or pruned model [14] instead. Since sub-models and pruned models are more lightweight than the global model, the amount of data in updates uploading is reduced.

Computation resources of edge devices are scarce compared to the cloud. Additional challenges should be considered to improve communication efficiencies: (1) Computation resources are heterogeneous and limited at edge devices; (2) Training data at edge devices may be distributed non-uniformly [11, 33, 34]. For more powerful edge devices, *ADSP* [10] lets them continue training while committing model aggregation at strategically decided intervals. For general cases, based on the deduced convergence bound for distributed learning with non-IID data distributions, the aggregation frequency under given resource budgets among all participating devices can be optimized with theoretical guarantees [11]. *Astraea* [13] reduces 92% communication traffic by designing a mediator-based multi-client rescheduling strategy. On the one hand, *Astraea* leverages data augmentation [24] to alleviate the defect of non-uniformly distributed training data. On the other hand, *Astraea* designs a greedy strategy for mediator-based rescheduling, in order to assign clients to the mediators. Each mediator traverses the data distribution of all unassigned clients to select the appropriate participating clients, aiming to make the mediator's data distribution closest to the uniform distribution, i.e., minimizing the Kullback–Leibler divergence [35] between mediator's data distribution and uniform distribution. When a mediator reaches the max assigned clients limitation, the central server will create a new mediator and repeat the process until all clients have been assigned with training tasks.

Aiming to accelerate the global aggregation in FL, [36] takes advantage of over-the-air computation [37–39], of which the principle is to explore the superposition property of a wireless multiple-access channel to compute the desired function by the concurrent transmission of multiple edge devices. The interferences of wireless channels can be harnessed instead of merely overcoming them. During

the transmission, concurrent analog signals from edge devices can be naturally weighed by channel coefficients. Then the server only needs to superpose these reshaped weights as the aggregation results, nonetheless, without other aggregation operations.

6.4 Resource-Optimized FL

When FL deploys the same neural network model to heterogeneous edge devices, devices with weak computing power (stragglers) may greatly delay the global model aggregation. Although the training model can be optimized to accelerate the stragglers, due to the limited resources of heterogeneous equipment, the optimized model usually leads to diverged structures and severely defect the collaborative convergence. *ELFISH* [15] first analyzes the computation consumption of the model training in terms of the time cost, memory usage, and computation workload. Under the guidance of the model analysis, which neurons need to be masked in each layer to ensure that the computation consumption of model training meets specific resource constraints can be determined. Second, unlike generating a deterministically optimized model with diverged structures, different sets of neurons will be dynamically masked in each training period and recovered and updated during the subsequent aggregation period, thereby ensuring comprehensive model updates overtime. It is worth noting that although *ELFISH* improves the training speed by $2\times$ through resource optimization, the idea of *ELFISH* is to make all stragglers work synchronously, the synchronous aggregation of which may not able to handle extreme situations.

When FL is deployed in a mobile edge computing scenario, the wall-clock time of FL will mainly depend on the number of clients and their computing capabilities. Specifically, the total wall-clock time of FL includes not only the computation time but also the communication time of all clients. On the one hand, the computation time of a client depends on the computing capability of the clients and local data sizes. On the other hand, the communication time correlates to clients' channel gains, transmission power, and local data sizes. Therefore, to minimize the wall-clock training time of the FL, appropriate resource allocation for the FL needs to consider not only FL parameters, such as accuracy level for computation–communication trade-off, but also the resources allocation on the client side, such as power and CPU cycles.

However, minimizing the energy consumption of the client and the FL wall-clock time are conflicting. For example, the client can save energy by always maintain its CPU at low frequency, but this will definitely increase training time. Therefore, in order to strike a balance between energy cost and training time, the authors of [16] first design a new FL algorithm *FEDL* for each client to solve its local problem approximately till a local accuracy level achieved. Then, by using Pareto efficiency model [40], they formulate a non-convex resource allocation problem for *FEDL* over wireless networks to capture the trade-off between the clients' energy cost and the

FL wall-clock time. Finally, by exploiting the special structure of that problem, they decompose it into three sub-problems, and accordingly derive closed-form solutions and characterize the impact of the Pareto-efficient controlling knob to the optimal.

Since the uplink bandwidth for transmitting model updates is limited, the BS must optimize its resource allocation while the user must optimize its transmit power allocation to reduce the packet error rates of each user, thereby improving FL performance. To this end, the authors of [17] formulate resource allocation and user selection of FL into a joint optimization problem, the goal of which is to minimize the value of the FL loss function while meeting the delay and energy consumption requirements. To solve this problem, they first derive a closed-form expression for the expected convergence rate of the FL in order to establish an explicit relationship between the packet error rates and the FL performance. Based on this relationship, the optimization problem can be reduced to a mixed-integer nonlinear programming problem, and then solved as follows: First, find the optimal transmit power under a given user selection and resource block allocation; Then, transform the original optimization problem into a binary matching problem; Finally, using Hungarian algorithm [41] to find the best user selection and resource block allocation strategy.

The number of devices involved in FL is usually large, ranging from hundreds to millions. Simply minimizing the average loss in such a large network may be not suited for the required model performance on some devices. In fact, although the average accuracy under vanilla FL is high, the model accuracy required for individual devices may not be guaranteed. To this end, based on the utility function α-fairness [42] used in fair resource allocation in wireless networks, the authors of [18] define a fair-oriented goal q-FFL for joint resource optimization. q-FFL minimizes an aggregate re-weighted loss parameterized by q, so that devices with higher loss are given higher relative weight, thus encouraging less variance (i.e., more fairness) in the accuracy distribution. Adaptively minimizing q-FFL avoids the burden of handcrafting fairness constraints, and can adjust the goal according to the required fairness dynamically, achieving the effect of reducing the variance of accuracy distribution among participated devices.

6.5 Security-Enhanced FL

Information interaction of participants in distributed training has always been a key point of privacy issues in distributed machine learning, which is briefly discussed in above Sect. 6.2. FL avoids possible privacy leakage caused by uploading training data, but meanwhile introduces privacy issues about model updates [24]. By adding noise to the sensitive data, Differential privacy (DP) controls the disclosure the information with rigorous and quantified expressions to ensure that any individual's data in or out of the whole dataset has almost no effect on the final result, and, which is helpful to reduce the threat of privacy disclosure in FL training updates (Fig. 6.3).

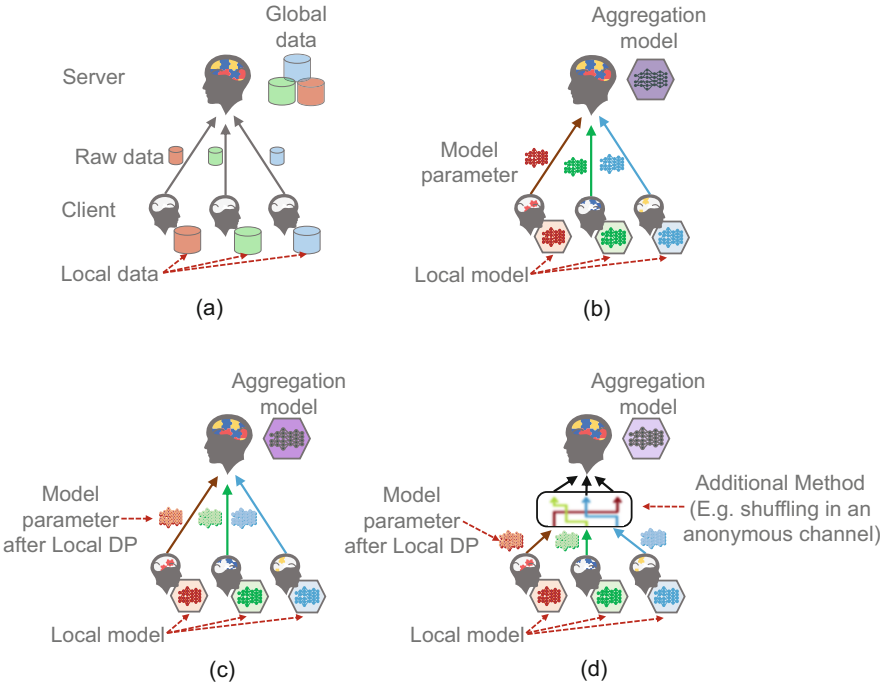

Fig. 6.3 The object and mechanism of privacy protection in different modes. (**a**) Centralized learning. (**b**) Federated learning (FL). (**c**) FL with local DP. (**d**) FL with distributed DP

Considering aggregation servers can receive model updates from all the participants and the servers locate in a hierarchical and heterogeneous computing environment with multi-domains, it is hard to provide the sufficient level of security anywhere with acceptable cost, which makes the servers a vulnerable point for privacy protection. To solve this issue, the idea of integrating local models of DP (local DP) into iterative training process does not have to trust the aggregation server, which can be summarized as follows: the participants first compute a model update with their own data, then process the update with DP, and finally upload the processed update for model aggregation. To give a general approach to allocate privacy budget and analyze the privacy cost in different training contexts (FL or not FL), a modular approach is proposed in [43] to separate the DP-incorporated training process into three aspects: (1) the specification of the training procedure, (2) the selection and configuration of the privacy mechanisms, and (3) the accounting procedure for the DP guarantee. However, the utility degradation of local DP in dealing with high-dimensional problem (e.g., the application in machine learning) is still challenging and attracts research efforts [44]. Furthermore, distributed models of DP (Distributed DP) can also protect the privacy of participants without relying on a trustworthy server and outperforms the same level of local DP on accuracy [45]. In distributed DP, communication costs need to be considered additionally based on

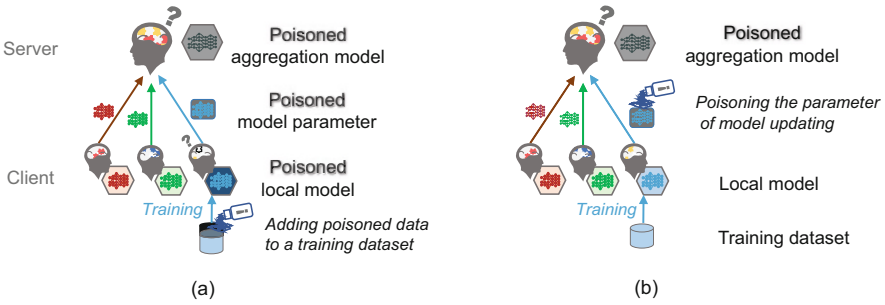

Fig. 6.4 Data poisoning attack and model update poisoning attack. (**a**) Data poisoning attack. (**b**) Model update poisoning attack

the trade-off between accuracy and privacy, and additional methods are typically introduced.

To look at it the other way, the server also should not trust training devices completely, since adversaries may be able to poison their training data or directly tamper with model updates, and hence resulting in damage to the global model. To make FL capable of tolerating a small number of devices training on the poisoned dataset, *robust federated optimization* [19] defines a trimmed mean operation. By filtering out not only the values produced by poisoned devices but also the natural outliers in the normal devices, robust aggregation protecting the global model from data poisoning is achieved. More directly, adversaries can compromise the integrity of update information. An attack methodology *model-replacement* is developed in [46] to enable adversaries to inject a backdoor through model update poisoning into the aggregation model, which shows more serious effects than data poisoning attack. The backdoor of training model is to degrade the performance on targeted sub-tasks without undue impact on overall performance. Concretely, the backdoor in FL is further studied in [47], which shows that (1) the successful introduction of the backdoor largely depends on the fraction of adversaries. (2) Norm clipping is effective in mitigating the effects of adversaries, and adding Gaussian noise based on norm clipping can further enhance the performance of the defense. To mitigate the impact which an adversary disguises multiple identities to introduce the backdoor, a defense method *FoolsGold* is proposed to distinguish abnormal updates by the diversity derived from non-IID training data and the similarity caused by the same attack objective [20] (Fig. 6.4).

Other than intentional attacks, passive adverse effects on the security, brought by unpredictable network conditions and computation capabilities, should be concerned as well. Wireless communication noise inevitably hinders the information exchange between training devices and the aggregation server, which may have a significant impact on training delay and model reliability. In [21], a parallel optimization problem under the expectation-based model and the worst-case model is formulated, and the two models are solved separately by the regularization for a loss function approximation algorithm and a sampling-based successive convex

approximation algorithm. The theoretical analysis shows the convergence properties with acceptable rates, and the simulation demonstrates improved model accuracy and reduced loss function. Beyond that, FL must be robust to the unexpectedly drop out of edge devices, or else once a device loses its connection, the synchronization of FL in one round will be failed. To solve this issue, *Secure Aggregation* protocol is proposed in [22] to achieve the robustness of tolerating up to one-third devices failing to timely process the local training or upload the updates.

In turn, malfunctions of the aggregation server in FL may result in inaccurate global model updates and thereby distorting all local model updates. Besides, edge devices with a larger number of data samples may be less willing to participate in FL with others that have less contribution. Therefore, in [23], combining Blockchain and FL as *BlockFL* is proposed to realize (1) locally global model updating at each edge device rather a specific server, ensuring device malfunction cannot affect other local updates when updating the global model; (2) appropriate reward mechanism for stimulating edge devices to participate in FL.

6.6 A Case Study for Training DRL at Edge

To further introduce DRL at edge, a practical case will be described in detail in this section. In the edge computing scenario, the distribution of resources is uneven. Generally speaking, computing tasks are generated at the end device, but the resources of the end device have limitations. In contrast, the resources on the edge will be more abundant, but transmitting tasks to the edge for execution will cause additional resource consumption and delay. Therefore, how to coordinate the resources of the end and the edge becomes a difficult problem. This section will use DRL to perform computational offloading to maximize performance.

6.6.1 Multi-User Edge Computing Scenario

First, define L_u to represent the set of all end devices in the scene, and the target device as $i_{\text{cur}} \in L_u$. The location of the target device is in the service range of some edge nodes, and its different distance from each edge node will affect the transmission rate. Define the set of edge nodes as $B = \{1, 2, \ldots, b\}$. Furthermore, the geographical location, task processing capability, and data transmission capability of each edge node are different. Besides, each edge node has multiple channels for end devices to connect. However, when there are too many end devices connected to a channel, the channel will be interfered and the data transmission rate will be reduced.

The edge computing scenario uses the idea of time slicing. The length of each time slice is δ, and the current time slice number is j. At the beginning of each time slice, the end device will generate tasks and obtain energy according to a certain

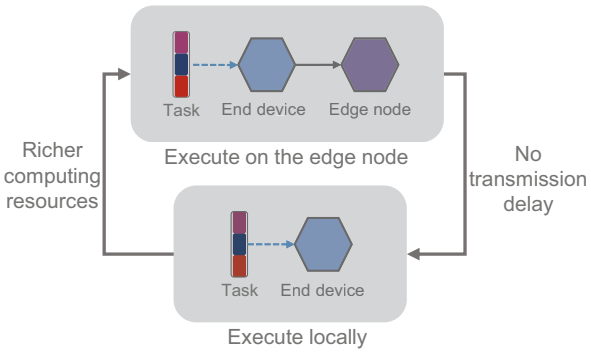

Fig. 6.5 Execution types of computing tasks

probability distribution. Further, define $a_j \in \{0, 1\}$ as a task generation indicator. When $a_j = 1$, a task is generated, otherwise it means no task is generated. The generated tasks will be stored in the task queue L_t^j. If the task queue cannot be stored at this time, the task will be lost and fail. Similarly, the energy obtained by the end device is also stored in the energy queue L_e^j.

Both task queue and energy queue are executed according to the principle of First In First Out (FIFO). The definitions of u and v, respectively, represent the amount of task data to be processed and the required CPU frequency. In addition, the action decision of the resource allocation strategy in epoch j is (c^j, e^j), where c^j is the task offloading decision and e^j represents the number of energy units to be allocated. If $e^j = 0$, the task will not be executed and will still be stored in the task queue. If $e^j > 0$, the task will be executed.

6.6.2 System Formulation

As shown in Fig. 6.5, there are two types of execution of computing tasks in edge computing scenarios. (1) Local execution on end devices. (2) Transfer the task to the edge node and have the edge node execute it.

When the task is executed locally, first calculate the CPU frequency f_u^j allocated by the end device by the Eq. 6.1:

$$f_u^j = \min \left\{ \sqrt{\frac{e^j}{v \cdot \tau}}, f_u^{\max} \right\}, \tag{6.1}$$

where $f_u^{\max}$ represents the maximum CPU frequency of the end device and τ is the effective switched capacitance determined by the hardware structure of the device.

Then as shown in the Eq. 6.2 , according to f_u^j , we can calculate the time d^j required to execute the task:

$$d^j = \frac{v}{f_u^j} \tag{6.2}$$

Because the edge node is not used when performing the computing task locally, it costs $\varphi^j = 0$ to occupy the edge node.

When a task needs to be transmitted to the edge node for execution, the time it takes for the task to execute is shown in the Eq. 6.3. When the edge node performs a task, the cost of occupying the edge node φ^j is shown in the Eq. 6.4, where P is the cost of using the edge node unit time.

$$d^j = d_c + d_{\text{tr}}^j + d_e^j \tag{6.3}$$

$$\varphi^j = \left(d_c + d_{\text{tr}}^j\right) \cdot P, \tag{6.4}$$

where d_c represents the time it takes to establish a channel connection and d_{tr}^j represents the time it takes to transfer the data required by the task to the edge node, which can be calculated by the Eqs. 6.6–6.8. d_e^j represents the execution time of the task at the edge node and can be calculated by the Eq. 6.5:

$$d_e^j = \frac{v}{f_e^j}, \tag{6.5}$$

where f_e^j represents the CPU frequency allocated by the edge node.

$$d_{\text{tr}}^j = \frac{u}{r_u}, \tag{6.6}$$

where r_u represents the transmission rate of task data and it can be derived from the Eq. 6.7:

$$r_u = W \cdot \log_2\left(1 + I^{-1} \cdot g_b^i \cdot p_{\text{tr}}^i\right), \tag{6.7}$$

where W represents the bandwidth of the selected channel and it is determined by the hardware equipment of the edge node. p_{tr}^i represents the transmission power when the user i transmits task data to the edge node, which can be derived by the Eq. 6.8. I represents the interference power of the current channel, which can be derived from the Eq. 6.9. The interference power of all channels is recorded as the set L_I^j and g_b^i represents the channel gain of user i at channel b.

$$p_{\text{tr}}^i = \min\left\{\frac{e^j}{g_{\text{tr}}^i}, p_{\text{tr}}^{\max}\right\}, \tag{6.8}$$

where $p_{\text{tr}}^{\max}$ represents the maximum transmission power and it is determined by the hardware device. Set all devices using the same channel as the target device to the set L_c, then

$$I = \sum_{i \in L_c} g_b^i \cdot p_{\text{tr}}^i - g_b^{i_{\text{cur}}} \cdot p_{\text{tr}}^{i_{\text{cur}}} \tag{6.9}$$

In addition, the network status is defined as $X_i = (L_t^j, L_e^j, L_l^j)$, and its definition initial value is X'. $\Phi(X_i)$ represents a policy decision function based on the environment X_i. The comprehensive reward U can be derived from the formula 6.10, where w_i is the weight of the target device's demand preference for different indicators, and the value is determined by the device type and the application service.

$$U(X_i, \Phi) = w_1 \cdot e^{-d^j} + w_2 \cdot e^{-e^j} + w_3 \cdot e^{-\eta^j} + w_4 \cdot e^{-\varphi^j} \tag{6.10}$$

The optimization goal of the strategy is to maximize the long-term comprehensive reward U^{long} as shown in the Eq. 6.11.

$$U^{\text{long}} = \max \left[\lim_{T \to +\infty} \frac{1}{T} \sum_{t=1}^{T} U(X_t, \Phi) | X_1 = X' \right]. \tag{6.11}$$

6.6.3 *Offloading Strategy for Computing Tasks Based on DRL*

By studying the rewards brought by different decisions, DRL can autonomously choose the corresponding best action for different environmental states, so it can achieve adaptive collaborative optimization of end devices and edge nodes in the edge computing scenario. Therefore, the offloading strategy in edge computing will be designed and implemented based on the algorithm idea of DQN, a representative algorithm in DRL. First, record the task queue length, energy queue length, and all channel states of each edge node in the scene as the observation value s in the environment. Then enter s into the current network of DQN, and select the action $a_{\max}$ according to the $\epsilon - greedy$ strategy, including the decision on the amount of energy allocation and the decision on task offload. After executing the decision and acting on the environment, four indicators of task delay, energy consumption, task failure rate, and occupation cost of the edge nodes can be obtained. Therefore, the comprehensive reward is calculated as r according to the different demand preferences of the end users. In addition, the observation value is recorded again as s' after the action is performed in the environment, and then $(s, a_{\max}, r, s')$ is stored in the replay buffer. When the sample D accumulated in the replay buffer is greater than the number of samples D_{batch} that needs to be extracted each time, it starts to randomly select some samples for learning and update the parameters. When the number of iterations of the above process $i \% N_{\text{update}} = 0$, the

parameters of the current network are copied to the target network. The parameters are continuously updated through iterations to achieve convergence eventually, so as to train a strategy that can make accurate decisions based on environmental.

6.6.4 Distributed Cooperative Training

In order to further strengthen the privacy protection of the data and reduce the data transmission, for the training of the above-mentioned DRL-based offloading strategy, a distributed collaborative training method can be adopted. First, an agent is initialized at both the end device and the edge node and the following process is then iterated:

(1) The end device d downloads the parameters θ_t from the edge node and assigns them to the local parameters θ_t^d of the end device, that is, $\theta_t^d = \theta_t$. The end device then updates the parameters $\theta_t^d = \theta_{t+1}^d$ through training based on local samples.
(2) The end device uploads the trained parameters θ_{t+1}^d to the edge node, and uploads the number of local training times of the end device A_t^d.
(3) The edge node updates the training times $A_t = \sum A_t^d$ by receiving data from the end device, and updates the parameter θ_{t+1} by aggregation, i.e., $\theta_{t+1} \leftarrow \sum \left(A_t^d / A_t\right) \cdot \theta_{t+1}^d$.

Through the above training process, distributed collaborative training can be achieved without directly transmitting sample data.

References

1. Y. Huang, Y. Zhu, X. Fan, et al., Task scheduling with optimized transmission time in collaborative cloud–edge learning, in *Proceedings of the 27th International Conference on Computer Communication and Networks (ICCCN 2018)* (2018), pp. 1–9
2. Why continuous learning is key to AI. Available: https://www.oreilly.com/radar/why-continuous-learning-is-key-to-ai/
3. G. Kamath, P. Agnihotri, M. Valero, et al., Pushing analytics to the edge, in *2016 IEEE Global Communications Conference (GLOBECOM 2016)* (2016), pp. 1–6
4. L. Valerio, A. Passarella, M. Conti, A communication efficient distributed learning framework for smart environments. Pervasive Mob. Comput. **41**, 46–68 (2017)
5. H.B. McMahan, E. Moore, D. Ramage, et al., Communication-efficient learning of deep networks from decentralized data, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)* (2017), pp. 1273–1282
6. K. Bonawitz, H. Eichner, et al., Towards federated learning at scale: System design (2019). Preprint. arXiv:1902.01046
7. M.S.H. Abad, E. Ozfatura, D. Gunduz, O. Ercetin, Hierarchical federated learning across heterogeneous cellular networks (2019). Preprint. arXiv: 1909.02362
8. J. Konečný, H.B. McMahan, F.X. Yu, et al., Federated learning: Strategies for improving communication efficiency (2016). Preprint. arXiv:1610.05492

9. S. Caldas, J. Konečny, H.B. McMahan, A. Talwalkar, Expanding the reach of federated learning by reducing client resource requirements (2018). Preprint. arXiv:1812.07210
10. H. Hu, D. Wang, C. Wu, Distributed machine learning through heterogeneous edge systems (2019). Preprint. arXiv:1911.06949
11. S. Wang, T. Tuor, T. Salonidis, et al., When edge meets learning: Adaptive control for resource-constrained distributed machine learning, in *IEEE Conference on Computer Communications (INFOCOM 2018)* (2018), pp. 63–71
12. A. Reisizadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, R. Pedarsani, FedPAQ: A Communication-efficient federated learning method with periodic averaging and quantization (2019). Preprint. arXiv:1909.13014
13. M. Duan, Astraea: Self-balancing federated learning for improving classification accuracy of mobile deep learning applications (2019). Preprint. arXiv:1907.01132
14. Y. Jiang, S. Wang, B.J. Ko, W.-H. Lee, L. Tassiulas, Model pruning enables efficient federated learning on edge devices (2019). Preprint. arXiv:1909.12326
15. Z. Xu, Z. Yang, J. Xiong, J. Yang, X. Chen, ELFISH: Resource-aware federated learning on heterogeneous edge devices (2019). Preprint. arXiv:1912.01684
16. C. Dinh, N.H. Tran, M.N.H. Nguyen, C.S. Hong, W. Bao, A.Y. Zomaya, V. Gramoli, Federated learning over wireless networks: Convergence analysis and resource allocation (2019). Preprint. arXiv:1910.13067
17. M. Chen, Z. Yang, W. Saad, C. Yin, H.V. Poor, S. Cui, A joint learning and communications framework for federated learning over wireless networks (2019). Preprint. arXiv:1909.07972
18. T. Li, M. Sanjabi, V. Smith, Fair resource allocation in federated learning (2019). Preprint. arXiv:1905.10497
19. C. Xie, S. Koyejo, I. Gupta, Practical distributed learning: Secure machine learning with communication-efficient local updates (2019). Preprint. arXiv:1903.06996
20. C. Fung, C.J.M. Yoon, I. Beschastnikh, Mitigating sybils in federated learning poisoning (2018). Preprint. arXiv:1808.04866
21. F. Ang, L. Chen, N. Zhao, et al., Robust federated learning with noisy communication (2019). Preprint. arXiv:1911.00251
22. K. Bonawitz, V. Ivanov, B. Kreuter, et al., Practical secure aggregation for privacy-preserving machine learning, in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS 2017)* (2017), pp. 1175–1191
23. H. Kim, J. Park, M. Bennis, S.-L. Kim, On-device federated learning via blockchain and its latency analysis (2018). Preprint. arXiv:1808.03949
24. Y. Lin, S. Han, H. Mao, et al., Deep gradient compression: reducing the communication bandwidth for distributed training (2017). eprint arXiv:1712.01887
25. Z. Tao, C. William, eSGD: Communication efficient distributed deep learning on the edge, in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge)* (2018), pp. 1–6
26. N. Strom, Scalable distributed DNN training using commodity GPU cloud computing, in *16th Annual Conference of the International Speech Communication Association (INTERSPEECH 2015)* (2015), pp. 1488–1492
27. E. Jeong, S. Oh, H. Kim, et al., Communication-efficient on-device machine learning: Federated distillation and augmentation under non-IID private data (2018). Preprint. arXiv:1811.11479.
28. M. Fredrikson, S. Jha, T. Ristenpart, Model inversion attacks that exploit confidence information and basic countermeasures, in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS 2015)* (2015), pp. 1322–1333
29. M. Du, K. Wang, Z. Xia, Y. Zhang, Differential privacy preserving of training model in wireless big data with edge computing. IEEE Trans. Big Data **6**, 283–295 (2018). (Early Access)
30. C. Dwork, F. McSherry, K. Nissim, A. Smith, Calibrating noise to sensitivity in private data analysis, in *Theory of Cryptography* (Springer, Berlin, 2006), pp. 265–284
31. S. Samarakoon, M. Bennis, W. Saad, M. Debbah, Distributed federated learning for ultra-reliable low-latency vehicular communications. IEEE Trans. Commun. **68**, 1146–1159 (2020, Early Access)

32. B.S. Kashin, Diameters of some finite-dimensional sets and classes of smooth functions. Izv. Akad. Nauk SSSR Ser. Mat. **41**, 334–351 (1977)
33. S. Wang, T. Tuor, T. Salonidis, et al., Adaptive federated learning in resource constrained edge computing systems. IEEE J. Sel. Areas Commun. **37**(6), 1205–1221 (2019)
34. T. Tuor, S. Wang, T. Salonidis, et al., Demo abstract: Distributed machine learning at resource-limited edge nodes, in *2018 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS 2018)* (2018), pp. 1–2
35. S. Kullback, R.A. Leibler, On information and sufficiency. Ann. Math. Stat. **22**(1), 79–86 (1951)
36. K. Yang, T. Jiang, Y. Shi, Z. Ding, Federated learning via over-the-air computation (2018). Preprint. arXiv:1812.11750
37. B. Nazer, et al., Computation over multiple-access channels. IEEE Trans. Inf. Theory **53**(10), 3498–3516 (2007)
38. L. Chen, N. Zhao, Y. Chen, et al., Over-the-air computation for IoT networks: computing multiple functions with antenna arrays. IEEE Internet Things J. **5**(6), 5296–5306 (2018)
39. G. Zhu, Y. Wang, K. Huang, Broadband analog aggregation for low-latency federated edge learning (extended version) (2018). Preprint. arXiv:1812.11494
40. J.E. Stiglitz, Self-selection and Pareto efficient taxation. J. Public Econ. **17**(2), 213–240 (1982)
41. H.W. Kuhn, The hungarian method for the assignment problem. Nav. Res. Logist. Q. **2**(1-Ř-2), 83–97 (1955)
42. H. SHI, R.V. Prasad, E. Onur, I.G.M.M. Niemegeers, Fairness in wireless networks:issues, measures and challenges. IEEE Commun. Surv. Tutor. **16**(1), 5–24 (First Quarter 2014)
43. H.B. McMahan, G. Andrew, U. Erlingsson, et al., A general approach to adding differential privacy to iterative training procedures (2018). Preprint. arXiv:1812.06210 (2018)
44. A. Bhowmick, J. Duchi, J. Freudiger, G. Kapoor, R. Rogers, Protection against reconstruction and its applications in private federated learning (2018). Preprint. arXiv:1812.00984
45. A. Cheu, A. Smith, J. Ullman, D. Zeber, M. Zhilyaev, Distributed differential privacy via shuffling (2018). Preprint. arXiv:1808.01394
46. E. Bagdasaryan, A. Veit, Y. Hua, D. Estrin, V. Shmatikov, How to backdoor federated learning (2018). Preprint. arXiv:1807.00459
47. Z. Sun, P. Kairouz, A.T. Suresh, H.B. McMahan, Can you really backdoor federated learning? (2019). Preprint. arXiv:1911.07963

Chapter 7
Edge Computing for Artificial Intelligence

Abstract Extensive deployment of AI services, especially mobile AI, requires the support of edge computing. This support is not just at the network architecture level, the design, adaptation, and optimization of edge hardware and software are equally important. Specifically, (1) customized edge hardware and corresponding optimized software frameworks and libraries can help AI execution more efficiently; (2) the edge computing architecture can enable the offloading of AI computation; (3) well-designed edge computing frameworks can better maintain AI services running on the edge; (4) fair platforms for evaluating Edge AI performance help further evolve the above implementations.

7.1 Edge Hardware for AI

7.1.1 Mobile CPUs and GPUs

AI applications are more valuable if directly enabled on lightweight edge devices, such as mobile phones, wearable devices, and surveillance cameras, near to the location of events. Low-power IoT edge devices can be used to undertake lightweight AI computation, and hence avoiding communication with the cloud, but it still faces several challenges: (1) limited computation resources; (2) limited memory footprint; (3) limited energy consumption bottleneck.

To break through these bottlenecks, in [1], the authors focus on ARM Cortex-M micro-controllers and develop *CMSIS-NN*, a collection of efficient NN kernels. By *CMSIS-NN*, the memory footprint of NNs on ARM Cortex-M processor cores can be minimized, and then the AI model can be fitted into IoT devices, meantime achieving normal performance and energy efficiency.

Nonetheless, it is impossible to fit a larger or deeper state-of-the-art AI model into the low-power device. Naturally, commodity CPUs on smartphones (noted that these CPUs are not exclusive for smartphones and are also available for other smart devices) can execute AI with higher performance. However, without the acceleration of a mobile GPU or other optimizations, intensive AI computation is still unaffordable. For instance, performing more powerful AI models with respect

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_7

to continuous vision applications, such as commonly used VGG [2] consisting of 3 FCNNs layers and 13 CNNs layers, may take 100 s to process a single image on Samsung Galaxy S7 [3].

With regard to the bottleneck when running CNN layers on mobile GPUs, *DeepMon*, presented in [3] as a suite of optimizations (on both hardware and software) for processing convolution layers on mobile GPUs, can largely reduce the inference time of VGG by taking advantage of the similarity between consecutive frames in first-person-view videos. Such a feature incurs the large amount of computation repetition in CNN layers, and, thus, inspires the idea of caching computation results of CNN layers. In addition, by means of matrix decomposition, high-dimensional matrix operations, particularly multiplications, in CNNs layers become available in mobile GPUs and can be accelerated. In view of this work, different kinds of mobile GPUs, already deployed in pervasive edge devices, can be potentially explored with specific AI models and play a more important role in enabling edge intelligence.

Other than AI inference [1, 3], important factors that affect the performance of AI training on mobile CPUs and GPUs are discussed in [4]. It preliminarily studies whether mobile CPUs and GPUs are feasible for AI training, and if feasible, the important factors that affect the training performance on them. Since commonly used AI models, such as VGG [2], are too large for the memory size of mainstream edge devices, a relatively small Mentee network [5] is adopted to evaluate AI training. Evaluation results point out that the size of AI models is crucial for training performance and the efficient fusion of mobile CPUs and GPUs is important for accelerating the training process.

In industry, many manufacturers have also introduced products for edge computing. For CPU, in 2016, Intel acquired Movidius, The flagship product of Movidius was Myriad, a chip that is purpose-built for processing images and video streams. It is positioned as a VPU—Vision Processing Unit—for its ability to deal with computer vision. After acquiring Movidius, Intel has packaged Myriad 2 in a USB thumb drive form factor, which is sold as a Neural Compute Stick (NCS). The best thing about NCS is that it works with both x86 and ARM devices. It can be easily plugged into an Intel NUC or a Raspberry Pi for running inferencing. It draws power from the host device without the need for an external power supply.

When it comes to GPUs, NVIDIA is an undisputed market leader. NVIDIA has built the Jetson family of GPUs specifically for the edge. In terms of programmability, they are 100% compatible with their enterprise data center counterparts. These GPUs have fewer cores and draw less power than the traditional GPUs powering the desktops and servers. Jetson Nano is the most recent addition to the Jetson family which comes with a 128 core GPU. Nano is the most affordable GPU module that NVIDIA has ever shipped. With the cooperation of Azure IoT Edge and AWS IoT Greengrass, NVIDIA Jetson is already the most popular edge computing platform.

7.1.2 FPGA-Based Solutions

Though GPU solutions are widely adopted in the cloud for AI training and inference, however, restricted by the tough power and cost budget in the edge, these solutions may not be available. Besides, edge nodes should be able to serve multiple AI computation requests at a time, and it makes simply using lightweight CPUs and GPUs impractical. Therefore, edge hardware based on Field Programmable Gate Array (FPGA) is explored to study their feasibility for Edge AI.

FPGA is an integrated circuit which can be "field" programmed to work as per the intended design. It means it can work as a microprocessor, or as an encryption unit, or graphics card, or even all these three at once. As implied by the name itself, the FPGA is field programmable. So, an FPGA working as a microprocessor can be reprogrammed to function as the graphics card in the field, as opposed to in the semiconductor foundries. The designs running on FPGAs are generally created using hardware description languages such as VHDL and Verilog. Due to its flexibility in hardware, FPGA has many advantages in field of latency, connectivity, and energy efficiency compared with GPU.

Particularly for CNNs, [1] gives a solution, tested by a prototype design with FPGA, for IoT devices to deal with image detection. First, unnecessary data movement is reduced to minimize data access, and hence achieving high energy efficiency for AI computation. Second, large kernel-sized computation is decomposed into multiple parallel small kernel-sized computations. Thirdly, to achieve minimum hardware design cost, pooling functions are separated into two categories, viz., max pooling and average pooling functions, while the max pooling is computed in parallel with convolution and the convolution engine is reused for average pooling functions. By this means, CNN acceleration with arbitrarily sized convolution and reconfigurable pooling, which are both useful in handling different CNN structures, can be realized.

For deploying RNNs, especially LSTM, on FPGA, [6] presented an implementation with respect to speech recognition, and it achieves 43x and 3x faster than Core i7 5930k CPU and Pascal Titan X GPU implementations, while achieving higher energy efficiency nonetheless. Specifically, this design starts with developing load balance-aware LSTM model pruning and automatic flow for dynamic-precision quantization of model weights. Then, to tackle the irregular computation pattern brought by model compression, hardware architecture is devised for directly accommodating the sparse model. Further, concerning the scheduling complexity of LSTM operations, an efficient scheduler is developed on FPGA.

Different from [1, 6], the design and setup of a FPGA-based edge computing system were proposed in [7]. It focus on developing an architecture for offloading AI computation from mobile devices to the edge FPGA platform rather than provide adaptation on the FPGA platform for AI models. On implementing the FPGA-based edge, a wireless router and a FPGA board are combined together with two main components deploying on them, viz., *Offload Manager* module and *Computation Offloading* module. Testing this preliminary system with typical vision applications,

Table 7.1 Comparison of solutions for edge nodes

Metrics	Preferred hardware	Analysis
Resource overhead	FPGA	FPGA can be optimized by customized designs
DL training	GPU	Floating point capabilities are better on GPU
DL inference	FPGA	FPGA can be customized for specific DL models
Interface scalability	FPGA	It is more free to implement interfaces on FPGAs
Space occupation	CPU/ FPGA	Lower power consumption of FPGA leads to smaller space occupation
Compatibility	CPU/GPU	CPUs and GPUs have more stable architecture
Development efforts	CPU/GPU	Toolchains and software libraries facilitate the practical development
Energy efficiency	FPGA	Customized designs can be optimized
Concurrency support	FPGA	FPGAs are suitable for stream process
Timing latency	FPGA	Timing on FPGAs can be an order of magnitude faster than GPUs

the FPGA-based edge shows its advantages, in terms of both energy consumption and hardware cost, over the GPU-based edge.

Nevertheless, it is still pended to determine whether FPGAs or GPUs/CPUs are more suitable for edge computing, as shown in Table 7.1. Elaborated experiments are performed in [8] to investigate the advantages of FPGAs over GPUs: (1) capable of providing workload insensitive throughput; (2) guaranteeing consistently high performance for high-concurrency AI computation; (3) better energy efficiency. However, the disadvantage of FPGAs lies in that developing efficient AI algorithms on FPGA is unfamiliar to most programmers. Although tools such as Xilinx SDSoC can greatly reduce the difficulty [7], at least for now, additional works are still required to transplant the state-of-the-art AI models, programmed for GPUs, into the FPGA platform.

7.1.3 TPU-Based Solutions

More recently, Google announced the availability of Edge TPU, a flavor of its TPU designed to run at the edge. Edge TPU complements the Cloud TPU by performing inferencing of trained models at the edge. According to Google, Edge TPU is a purpose-built ASIC designed to run AI at the edge. It delivers high performance in a small physical and power footprint, enabling the deployment of high-accuracy AI at the edge. These purpose-built chips can be used for emerging use cases such as predictive maintenance, anomaly detection, machine vision, robotics, voice recognition, and many more.

Developers can optimize TensorFlow models for Edge TPU by converting them to TensorFlow Lite models compatible with the edge. Google has made a web-based

and command line tool to convert existing TensorFlow models optimized for Edge TPU. Unlike NVIDIA and Intel edge platforms, Google's Edge TPU cannot run models other than TensorFlow at the edge. Google is building a seamless pipeline that automates and simplifies the workflow involved in training the models in the cloud and deploying them on Edge TPU. Cloud AutoML Vision, an automated, no-code environment to train CNNs supports exporting the model into TensorFlow Lite format optimized for the Edge TPU. By far, this is the simplest workflow available to build cloud-to-edge pipelines.

In the future, with AI becoming the key driver of the edge, the combination of hardware accelerators and software platforms is becoming important to run the models for inference, with the release of more powerful edge AI chips and optimized AI frameworks for edge platforms, the way of AI training and inference may potentially be changed. With the continuous upgrade of hardware, more and more AI applications will be able to run in the Edge and make our life better.

7.2 Edge Data Analysis for Edge AI

As we all know, data is an important part of AI training and inference. Great data can help AI models to be trained faster and better. At the same time, fast and effective edge data processing architecture and algorithms can help the AI model achieve fast and accurate inference, so it can support more real-time AI applications. However, as the amount of data increases, especially in some real-time edge scenarios, it is not realistic to just use ordinary methods to send data to the AI model. Big data methods can be utilized to maximize the capabilities of edge + AI.

7.2.1 Challenge and Needs for Edge Data Process

There are several characteristics of the data that needs to be processed at the edge.

The first one is the large amount of data and the fast data rate. With an increase in the number of IoT devices, there has been a huge opportunity to use the extra data collected to bring even greater insight into our life. But perhaps there is now too much data. By 2020, more than 50 billion devices are expected to connect to the Internet, generating more than 400 ZB of data; and 70% of this data will need to be processed at the edge [9].

The second one is that there are too much useless data. Cisco has estimated that, driven by IoT devices, we will generate a staggering 847 ZB of data per year by 2021. However, useful data will only be around 85 ZB—just 10% of the total generated. Simply dealing with this overwhelming level of data using cloud systems will require ever-more expensive infrastructure.

Some data have higher requirements for real-time processing. For example, the video service provided for the wearable camera needs to be between 25 and 50 ms;

in some industrial scenarios like system monitoring, control, and execution, the real-time requirement is within 10 ms [10].

The rise of various smart applications in modern times like Autopilot, Drone, Smart Home, AR/VR, etc., has increased the diversity of edge source services like real-time video process, real-time recommendation, environmental awareness, etc. This requires that the edge processing architecture needs to be able to meet a variety of data processing needs.

7.2.2 Combination of Big Data and Edge Data Process

In order to make full use of the data and to mine useful information, we can combine the Edge Data Process with the Big Data technology. The Edge Data Process technology can be used to improve data collection, filter data, and provide real-time data process.

Cloud-based big data analysis is incredibly powerful and the more useful information you can give a system, the better the answers you can get from the questions you ask. But the difficulty is that in some scenarios, IoT devices are generating a staggering amount of data and not all of it is needed. Take demographics information, for example. With a cloud-based system, IoT cameras would have to collect video, send it to a central server, and then extract the necessary information.

With an Edge Computing solution, a computer connected to a camera can automatically strip out demographic information, sending that to the cloud for storage and processing. This dramatically cuts down on the amount of data collected, providing purely useful information. Likewise, with IoT sensors, is it necessary to send measurements every second for storage? By storing data locally and, perhaps, using averages for larger time periods, Edge Computing can cut down on noise, filtering data to provide only useful and relevant information. Most importantly, perhaps, in an age where people are worried about security and privacy, Edge Computing offers a responsible and secure way to collect data. Again, turning to our demographic information, no private video or facial data is sent to a server, rather an Edge Computer can take the useful, non-personalized data and transmit this to the cloud.

Edge AI can also be used for real-time big data analysis. For example, with facial recognition and demographics, a retail store could customize a digital display to show an offer that is likely to appeal to the person looking at it. Sending the video stream to the cloud, processing it and then displaying the right offer is too time-consuming. Using Edge Computing, a local computer can decode a person's demographic information and then display the appropriate offer in a fraction of the time. More than this, Edge AI devices can constantly monitor, and then take the right appropriate action, whether that is scaling back the speed of a production line to prevent damage, sending an alert, or even bringing a backup system online. The

key, again, is the speed of reaction, with Edge AI devices able to make real-time decisions, and all without requiring a fast internet connection.

7.2.3 Architecture for Edge Data Process

So far, the edge stream processing architecture can be divided into two types: Edge-based and Cloud–Edge based.

The Edge-based architecture uses edge devices or small edge clusters for data processing. In [11], the proposed architecture *F-Mstorm* achieves edge data process by enhancing streaming data process capabilities of edge devices. By using local mobile devices, it can process stream data in real time. It also support dynamic configuration and scheduling. In [12], the proposed infrastructure *Seagull* ties together small local servers and make them work together. Through effective scheduling algorithms and strategies, it can assign tasks to the appropriate nodes based on how close the nodes are to the sensor data source and how much processing the nodes can handle.

The Cloud–Edge based architecture combines edge devices or clusters with cloud data centers to provide a unified operating environment for collaborative processing of streaming data. On the one hand, in [13], authors propose *SpanEdge*, it reduces the network transmission delay by collaboratively optimizing the cloud and edge geographic distribution. On the other hand, some researchers put their focus more on better mapping of operators to physical nodes so that they can speed up data processing and sharing process [14].

7.3 Communication and Computation Modes for Edge AI

Although on-device AI computing illustrated in Sect. 5 can cater for lightweight AI services. Nevertheless, an independent end device still cannot afford intensive AI computation tasks. The concept of edge computing can potentially cope with this dilemma by offloading AI computation from end devices to edge or (and) the cloud. Accompanied by the edge architectures, AI-centric edge nodes can become the significant extension of cloud computing infrastructure to deal with massive AI tasks. In this section, we classify four modes for Edge AI computation, as exhibited in Fig. 7.1.

7.3.1 Integral Offloading

The most natural mode of AI computation offloading is similar to the existed "end–cloud" computing, i.e., the end device sends its computation requests to the

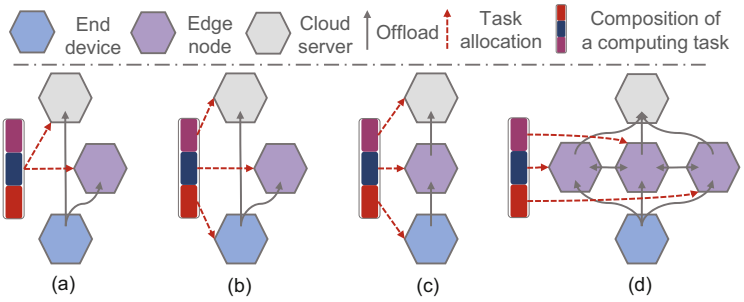

Fig. 7.1 Communication and computation modes for Edge AI. (**a**) Integral offloading. (**b**) Partial offloading. (**c**) Vertical collaboration. (**d**) Horizontal collaboration

cloud for AI inference results (as depicted in Fig. 7.1a). This kind of offloading is straightforward by extricating itself from AI task decomposition and combinatorial problems of resource optimization, which may bring about additional computation cost and scheduling delay, and thus simple to implement. In [15], the proposed distributed infrastructure *DeepDecision* ties together powerful edge nodes with less powerful end devices. In *DeepDecision*, AI inference can be performed on the end or the edge, depending on the trade-offs between the inference accuracy, the inference latency, the AI model size, the battery level, and network conditions. With regard to each AI task, the end device decides whether locally processing or offloading it to an edge node (Table 7.2).

Further, the workload optimization among edge nodes should not be ignored in the offloading problem, since edge nodes are commonly resource-restrained compared to the cloud. In order to satisfy the delay and energy requirements of accomplishing a AI task with limited edge resources, providing AI models with different model sizes and performance in the edge can be adopted to fulfill one kind of task. Hence, multiple VMs or containers, undertaking different AI models separately, can be deployed on the edge node to process AI requests. Specifically, when a AI model with lower complexity can meet the requirements, it is selected as the serving model. For instance, by optimizing the workload assignment weights and computing capacities of VMs, *MASM* [16] can reduce the energy cost and delay while guaranteeing the AI inference accuracy.

7.3.2 Partial Offloading

In Sect. 7.3.1, the strategy of offloading the integral AI computation task to the edge is concerned. Nonetheless, partially offloading the AI task to the edge is also feasible (as depicted in Fig. 7.1b). An offloading system can be developed to enable online fine-grained partition of a AI task, and determine how to allocate these divided tasks to the end device and the edge node. As exemplified in [20],

Table 7.2 Details about integral offloading modes for DL

Ref.	DL model	End/edge/cloud	Network	Dependency	Objective	Performance
Integral offloading						
DeepDecision [15]	YOLO	Samsung Galaxy S7/Server with a quad-core CPU at 2.7 GHz, GTX970 and 8 GB RAM/ N/A	Simulated WLAN and LAN	TensorFlow, Darknet	Consider the complex interaction between model accuracy, video quality, battery constraints, network data usage, and network conditions to determine an optimal offloading strategy	Achieve about 15 FPS video analytic while possessing higher accuracy than that of the baseline approaches
MASM [16]	\	Simulated devices/Cloudlet/ N/A	\	\	Optimize workload assignment weights and the computation capacities of the VMs hosted on the Cloudlet	\
EdgeEye [17]	DetectNet, FaceNet	Cameras/Server with Intel i7-6700, GTX 1060 and 24 GB RAM/ N/A	Wi-Fi	TensorRT, ParaDrop, Kurento	Offload the live video analytics tasks to the edge using EdgeEye API, instead of using DL framework specific APIs, to provide higher inference performance	\

MAUI, capable of adaptively partitioning general computer programs, can conserve an order of magnitude energy by optimizing the task allocation strategies, under the network constraints. More importantly, this solution can decompose the whole program at runtime instead of manually partitioning of programmers before program deploying. Though this work does not consider AI applications, it still sheds light on the potential of partial offloading of AI tasks (Table 7.3).

Further, particularly for AI computation, *DeepWear* [18] abstracts a AI model as a Directed Acyclic Graph (DAG), where each node represents a layer and each edge represents the data flow among those layers. To efficiently determine partial offloading decisions, *DeepWear* first prunes the DAG by keeping only the computation-intensive nodes, and then grouping the repeated sub-DAGs. In this manner, the complex DAG can be transformed into a linear and much simpler one, thus enabling a linear complexity solution for selecting the optimal partition to offload.

Nevertheless, uploading a part of the AI model to the edge nodes may still seriously delay the whole process of offloading AI computation. Some researchers point out it is not realistic to pre-install AI models on any edge nodes for handling every kind of requests from end devices, since which edge node will be used at runtime is hard to determine (especially when the mobility of devices is concerned) and it is even impractical to stuff all AI models into every edge node. Therefore, when the AI model is not pre-installed, the end device should first upload its AI model to the edge node. Unfortunately, it can seriously delay the offloaded AI computation due to long uploading time.

To deal with this challenge, an incremental offloading system *IONN* is proposed in [19]. Differ from packing up the whole AI model for uploading, *IONN* divides a AI model, prepared for uploading, into multiple partitions, and uploads them to the edge node in sequential. The edge node, receiving the partitioned models, incrementally builds the AI model as each partitioned model arrives, while being able to execute the offloaded partial AI computation even before the entire AI model is uploaded. Therefore, the key lies in the determination concerning the best partitions of the AI model and the uploading order. Specifically, on the one hand, DNN layers, performance benefit and uploading overhead of which are high and low, respectively, are preferred to be uploaded first, and, thus, making the edge node quickly build a partial DNN to achieve the best-expected query performance. On the other hand, unnecessary DNN layers, which cannot bring in any performance increase, are not uploaded and hence avoiding the offloading.

7.3.3 Vertical Collaboration

Expected offloading strategies among "End–Edge" architecture, as discussed in Sects. 7.3.1 and 7.3.2, are feasible for supporting less computation-intensive AI services and small-scale concurrent AI queries. However, when a large number of AI queries need to be processed at one time, a single edge node is certainly insufficient.

Table 7.3 Details about partial offloading modes for DL

Ref.	DL model	End/edge/cloud	Network	Dependency	Objective	Performance
Partial offloading						
DeepWear [18]	MobileNet, GoogLeNet, DeepSense, etc.	Commodity smartwatches running Android Wear OS/Commodity smartphone running Android/N/A	Bluetooth	TensorFlow	Provide context-aware offloading, strategic model partition, and pipelining support to efficiently utilize the processing capacity of the edge	Bring up to 5.08× and 23.0× execution speedup, as well as 53.5 and 85.5% energy saving against wearable-only and handheld-only strategies, respectively
IONN [19]	AlexNet	Embedded board Odroid XU4/Server with an quad-core CPU at 3.6 GHz, GTX 1080 Ti and 32 GB RAM/Unspecified	WLAN	Caffe	Partitions the DNN layers and incrementally uploads the partitions to allow collaborative execution by the end and the edge (or cloud) to improves both the query performance and the energy consumption	Maintain almost the same uploading latency as integral uploading while largely improving query execution time

a single edge node is certainly impractical to sustain them. Therefore, collaborative "End–Edge–Cloud" computing, as introduced in Sect. 2.1.5, can be utilized to deal with dozens of computation-intensive AI tasks (Table 7.4).

A natural choice of collaboration is the edge performs data preprocessing and preliminary learning, when the AI tasks are offloaded. Then, the intermediate data, viz., the output of edge architectures, are transmitted to the cloud for further AI computation [21]. Nevertheless, the hierarchical structure of DNNs can be further excavated for fitting the vertical collaboration. In [22], all layers of a DNN are profiled on the end device and the edge node in terms of the data and computation characteristics, in order to generate performance prediction models. Based on these prediction models, wireless conditions, and server load levels, the proposed *Neurosurgeon* evaluates each candidate point in terms of end-to-end latency or mobile energy consumption and partition the DNN at the best one. Then, it decides the allocation of DNN partitions, i.e., which part should be deployed on the end, the edge, or the cloud, while achieving best latency and energy consumption of end devices.

By taking advantages of EEoI (Sect. 5.3), vertical collaboration can be more adapted. Partitions of a DNN can be mapped onto a distributed computing hierarchy (i.e., the end, the edge, and the cloud) and can be trained with multiple early exit points [23]. Therefore, the end and the edge can perform a portion of AI inference on themselves rather than directly requesting the cloud. Using an exit point after inference, results of AI tasks, the local device is confident about, can be given without sending any information to the cloud. For providing more accurate AI inference, the intermediate DNN output will be sent to the cloud for further inference by using additional DNN layers. Nevertheless, the intermediate output, e.g., high-resolution surveillance video streams, should be carefully designed much smaller than the raw input, therefore, drastically reducing the network traffic required between the end and the edge (or the edge and the cloud).

Though vertical collaboration can be considered as an evolution of cloud computing, i.e., "end–cloud" strategy. Compared to the pure "end–edge" strategy, the process of vertical collaboration may possibly be delayed, due to it requires additional communication with the cloud. However, vertical collaboration has its own advantages. One side, when edge architectures cannot afford the flood of AI queries by themselves, the cloud architectures can share partial computation tasks and hence ensure servicing these queries. On the other hand, the raw data must be preprocessed at the edge before they are transmitted to the cloud. If these operations can largely reduce the size of intermediate data and hence reduce the network traffic, the pressure of backbone networks can be alleviated.

7.3.4 Horizontal Collaboration

In Sect. 7.3.3, vertical collaboration is discussed. However, devices among the edge or the end can also be united without the cloud to process resource-hungry AI

Table 7.4 Details about vertical collaboration modes for DL

Ref.	DL model	End/edge/cloud	Network	Dependency	Objective	Performance
Vertical collaboration						
[21]	CNN, LSTM	Google Nexus 9/Server with an quad-core CPU and 16 GB RAM/3 desktops, each with i7-6850K and 2×GTX 1080 Ti	WLAN and LAN	Apache Spark, TensorFlow	Perform data preprocessing and preliminary learning at the edge to reduce the network traffic, so as to speed up the computation in the cloud	Achieve 90% accuracy while reducing the execution time and the data transmission
Neurosurgeon [22]	AlexNet, VGG, Deepface, MNIST, Kaldi, SENNA	Jetson TK1 mobile platform/Server with Intel Xeon E5×2, NVIDIA Tesla K40 GPU and 256 GB RAM/Unspecified	Wi-Fi, LTE & 3G	Caffe	Adapt to various DNN architectures, hardware platforms, wireless connections, and server load levels, and choose the partition point for best latency and best mobile energy consumption	Improve end-to-end latency by 3.1× on average and up to 40.7×, reduce mobile energy consumption by 59.5% on average and up to 94.7%, and improve data-center throughput by 1.5× on average and up to 6.7×
[23]	BranchyNet	\	\	\	Minimize communication and resource usage for devices while allowing low-latency classification via EEoI	Reduce the communication cost by a factor of over 20× while achieving 95% overall accuracy
[24]	Faster R-CNN	Xiaomi 6/Server with i7 6700, GTX 980Ti and 32 GB RAM/Work station with E5-2683 V3, GTX TitanXp×4 and 128 GB RAM	WLAN and LAN	\	Achieve efficient object detection via wireless communications by interactions between the end, the edge and the cloud	Lose only 2.5% detection accuracy under the image compression ratio of 60% while significantly improving image transmission efficiency
VideoEdge [25]	AlexNet, DeepFace, VGG16	10 Azure nodes emulating Cameras/2 Azure nodes/12 Azure nodes	Emulated hierarchical networks	\	Introduce dominant demand to identify the best trade-off between multiple resources and accuracy	Improve accuracy by 5.4× compared to VideoStorm and only lose 6% accuracy of the optimum

applications, i.e., horizontal collaboration. By this means, the trained DNN models or the whole AI task can be partitioned and allocated to multiple end devices or edge nodes to accelerate AI computation by alleviating the resource cost of each of them. *MoDNN*, proposed in [26], executes AI in a local distributed mobile computing system over a Wireless Local Area Network (WLAN). Each layer of DNNs is partitioned into slices to increase parallelism and to reduce memory footprint, and these slices are executed layer-by-layer. By the execution parallelism among multiple end devices, the AI computation can be significantly accelerated (Table 7.5).

With regard to specific DNN structures, e.g., CNN, a finer grid partitioning can be applied to minimize communication, synchronization, and memory overhead [27]. In [28], a Fused Tile Partitioning (FTP) method, able to divide each CNN layer into independently distributable tasks, is proposed. In contrast to only partitioning the DNN by layers as in [22], FTP can fuse layers and partitions them vertically in a grid fashion, hence minimizing the required memory footprint of participated edge devices regardless of the number of partitions and devices, while reducing communication and task migration cost as well. Besides, to support FTP, a distributed work-stealing runtime system, viz., idle edge devices stealing tasks from other devices with active work items [28], can adaptively distribute FTP partitions for balancing the workload of collaborated edge devices.

7.4 Tailoring Edge Frameworks for AI

Though there are gaps between the computational complexity and energy efficiency required by AI and the capacity of edge hardware [31], customized Edge AI frameworks can help efficiently (1) match edge platform and AI models; (2) exploit underlying hardware in terms of performance and power; (3) orchestrate and maintain AI services automatically.

First, where to deploy AI services in edge computing (cellular) networks should be determined. The RAN controllers deployed at edge nodes are introduced in [32] to collect the data and run AI services, while the network controller, placed in the cloud, orchestrates the operations of the RAN controllers. In this manner, after running and feeding analytics and extract relevant metrics to AI models, these controllers can provide AI services to the users at the network edge.

Second, as the deployment environment and requirements of AI models can be substantially different from those during model development, customized operators, adopted in developing AI models with (Py)Torch, TensorFlow, etc., may not be directly executed with the AI framework at the edge. To bridge the gap between deployment and development, the authors of [33] propose to specify AI models in development using the deployment tool with an operator library from the AI framework deployed at the edge. Furthermore, to automate the selection and optimization of AI models, *ALOHA* [34] formulates a toolflow: (1) Automate the model design. It generates the optimal model configuration by taking into account the

Table 7.5 Details about horizontal collaboration modes for DL

Ref.	DL model	End/edge/cloud	Network	Dependency	Objective	Performance
Horizontal collaboration						
MoDNN [26]	VGG-16	Multiple LG Nexus 5/N/A/N/A	WLAN	MXNet	Partition already trained DNN models onto several mobile devices to accelerate DNN computations by alleviating device-level computing cost and memory usage	When the number of worker nodes increases from 2 to 4, MoDNN can speed up the DNN computation by 2.17–4.28×
[27]	VGGNet-E, AlexNet	Xilinx Virtex-7 FPGA simulating multiple end devices/N/A/N/A	On-chip simulation	Torch, Vivado HLS	Fuse the processing of multiple CNN layers and enable caching of intermediate data to save data transfer (bandwidth)	Reduce the total data transfer by 95%, from 77 MB down to 3.6 MB per image
DeepThings [28]	YOLOv2	Performance-limited Raspberry Pi 3 Model B/Raspberry Pi 3 Model B as gateway/N/A	WLAN	Darknet	Employ a scalable Fused Tile Partitioning of CNN layers to minimize memory footprint while exposing parallelism and a novel work scheduling process to reduce overall execution latency	Reduce memory footprint by more than 68% without sacrificing accuracy, improve throughput by 1.7×–2.2× and speed up CNN inference by 1.7×–3.5×
DeepCham [29]	AlexNet	Multiple LG G2/Wi-Fi router connected with a Linux server/N/A	WLAN and LAN	Android Caffe, OpenCV, EdgeBoxes	Coordinate participating mobile users for collaboratively training a domain-aware adaptation model to improve object recognition accuracy	Improve the object recognition accuracy by 150% when compared to that achieved merely using a generic DL model
LAVEA [30]	OpenALPR	Raspberry Pi 2 & Raspberry Pi 3/Servers with quad-core CPU and 4 GB RAM/N/A	WLAN and LAN	Docker, Redis	Design various task placement schemes that are tailed for inter-edge collaboration to minimize the service response time	Have a speedup ranging from 1.3× to 4× (1.2× to 1.7×) against running in local (client-cloud configuration)

target task, the set of constraints, and the target architecture; (2) Optimize the model configuration. It partitions the AI model and accordingly generates architecture-aware mapping information between different inference tasks and the available resources. (3) Automate the model porting. It translates the mapping information into adequate calls to computing and communication primitives exposed by the target architecture.

Third, the orchestration of AI models deployed at the edge should be addressed. *OpenEI* [35] defines each AI algorithm as a four-element tuple <Accuracy, Latency, Energy, Memory Footprint> to evaluate the Edge AI capability of the target hardware platform. Based on such tuple, *OpenEI* can select a matched model for a specific edge platform based on different Edge AI capabilities in an online manner. *Zoo* [36] provides a concise Domain-specific Language (DSL) to enable easy and type-safe composition of AI services. Besides, to enable a wide range of geographically distributed topologies, analytic engines, and AI services, *ECO* [37] uses a graph-based overlay network approach to (1) model and track pipelines and dependencies and then (2) map them to geographically distributed analytic engines ranging from small edge-based engines to powerful multi-node cloud-based engines. By this means, AI computation can be distributed as needed to manage cost and performance, while also supporting other practical situations, such as engine heterogeneity and discontinuous operations.

Nevertheless, these pioneer works are not ready to natively support valuable and also challenging features discussed in Sect. 7.3, such as computation offloading and collaboration, which still calls for further development.

7.5 Performance Evaluation for Edge AI

Throughout the process of selecting appropriate edge hardware and associated software stacks for deploying different kinds of Edge AI services, it is necessary to evaluate their performance. Impartial evaluation methodologies can point out possible directions to optimize software stacks for specific edge hardware.

In [38], for the first time, the performance of AI libraries is evaluated by executing AI inference on resource-constrained edge devices, pertaining to metrics like latency, memory footprint, and energy. In addition, particularly for Android smartphones, as one kind of edge devices with mobile CPUs or GPUs, *AI Benchmark* [39] extensively evaluates AI computation capabilities over various device configurations. Experimental results show that no single AI library or hardware platform can entirely outperform others, and loading the AI model may take more time than that of executing it. These discoveries imply that there are still opportunities to further optimize the fusion of edge hardware, edge software stacks, and AI libraries.

Nonetheless, a standard testbed for Edge AI is missing, which hinders the study of edge architectures for AI. To evaluate the end-to-end performance of Edge AI services, not only the edge computing architecture but also its combination

with end devices and the cloud shall be established, such as *openLEON* [40] and *CAVBench* [41] particularly for vehicular scenarios. Furthermore, simulations of the control panel of managing AI services are still not dabbled. An integrated testbed, consisting of wireless links and networking models, service requesting simulation, edge computing platforms, cloud architectures, etc., is ponderable in facilitating the evolution of "Edge Computing for AI."

References

1. L. Du et al., A reconfigurable streaming deep convolutional neural network accelerator for Internet of Things. IEEE Trans. Circuits Syst. Regul. Pap. **65**(1), 198–208 (2018)
2. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition (2014). Preprint. arXiv:1409.1556
3. L.N. Huynh, Y. Lee, R.K. Balan, DeepMon: mobile GPU-based deep learning framework for continuous vision applications, in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2017)* (2017), pp. 82–95
4. Y. Chen, S. Biookaghazadeh, M. Zhao, Exploring the capabilities of mobile devices supporting deep learning, in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2018)* (2018), pp. 17–18
5. R. Venkatesan, B. Li, Diving deeper into mentee networks (2016). Preprint. arXiv:1604.08220
6. S. Han, Y. Wang, H. Yang et al., ESE: efficient speech recognition engine with sparse LSTM on FPGA, in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017)* (2017), pp. 75–84
7. S. Jiang, D. He, C. Yang et al., Accelerating mobile applications at the network edge with software-programmable FPGAs, in *2018 IEEE Conference on Computer Communications (INFOCOM 2018)* (2018), pp. 55–62
8. S. Biookaghazadeh, F. Ren, M. Zhao, Are FPGAs suitable for edge computing? (2018). Preprint. arXiv:1804.06404
9. D. McAuley, R. Mortier, J. Goulding, The dataware manifesto, in *2011 Third International Conference on Communication Systems and Networks (COMSNETS 2011)* (2011), pp. 1–6
10. S. Agarwal, M. Philipose, P. Bahl, Vision: the case for cellular small cells for cloudlets, in *Proceedings of The International Workshop on Mobile Cloud Computing & Services* (2014), pp. 1–5
11. M. Chao, C. Yang, Y. Zeng, R. Stoleru, F-Mstorm: feedback-based online distributed mobile stream processing, in *2018 Third ACM/IEEE Symposium on Edge Computing* (2018), pp. 273–285
12. R.B. Das, G. Di Bernardo, H. Bal, Large stream analytics using a resource-constrained edge, in *2018 IEEE International Conference on Edge Computing* (2018), pp. 135–139
13. H.P. Sajjad, K. Danniswara, A. Al-Shishtawy, V. Vlassov, SpanEdge: towards unifying stream processing over central and near-the-edge data centers, in *2016 IEEE/ACM Symposium on Edge Computing (SEC)* (2016), pp. 168–178
14. Q. Zhang, Q. Zhang, W. Shi, H. Zhong, Firework: data processing and sharing for hybrid cloud-edge analytics. IEEE Trans. Parallel Distrib. Syst. **29**(9), 2004–2017 (2018)
15. X. Ran, H. Chen, X. Zhu, Z. Liu, J. Chen, DeepDecision: a mobile deep learning framework for edge video analytics, in *2018 IEEE Conference on Computer Communications (INFOCOM 2018)* (2018), pp. 1421–1429
16. W. Zhang, Z. Zhang, S. Zeadally et al., MASM: a multiple-algorithm service model for energy-delay optimization in edge artificial intelligence. IEEE Trans. Ind. Inf. **15**, 4216–4224 (2019)

17. P. Liu, B. Qi, S. Banerjee, EdgeEye – an edge service framework for real-time intelligent video analytics, in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2018)* (2018), pp. 1–6
18. M. Xu, F. Qian, M. Zhu, F. Huang, S. Pushp, X. Liu, DeepWear: adaptive local offloading for on-wearable deep learning. IEEE Trans. Mob. Comput. **19**, 314–330 (2019)
19. H.-j. Jeong, H.-j. Lee, C.H. Shin, S.-M. Moon, IONN: incremental offloading of neural network computations from mobile devices to edge servers, in *Proceedings of the ACM Symposium on Cloud Computing (SoCC 2018)* (2018), pp. 401–411
20. E. Cuervo, A. Balasubramanian, D.-k. Cho et al., MAUI: making smartphones last longer with code offload, in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services (MobiSys 2010)* (2010), pp. 49–62
21. Y. Huang, X. Ma, X. Fan et al., When deep learning meets edge computing, in *IEEE 25th International Conference on Network Protocols (ICNP 2017)* (2017), pp. 1–2
22. Y. Kang, J. Hauswald, C. Gao et al., Neurosurgeon: collaborative intelligence between the cloud and mobile edge, in *Proceedings of 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017)* (2017), pp. 615–629
23. S. Teerapittayanon, B. McDanel, H.T. Kung, Distributed deep neural networks over the cloud, the edge and end devices, in *IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017)* (2017), pp. 328–339
24. J. Ren, Y. Guo, D. Zhang et al., Distributed and efficient object detection in edge computing: challenges and solutions. IEEE Netw. **32**(6), 137–143 (2018)
25. C.-C. Hung, G. Ananthanarayanan, P. Bodik, L. Golubchik, M. Yu, P. Bahl, M. Philipose, VideoEdge: processing camera streams using hierarchical clusters, in *Proceedings of 2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)* (2018), pp. 115–131
26. J. Mao, X. Chen, K.W. Nixon et al., MoDNN: local distributed mobile computing system for Deep Neural Network, in *Design, Automation & Test in Europe Conference & Exhibition (DATE 2017)* (2017), pp. 1396–1401
27. M. Alwani, H. Chen, M. Ferdman, P. Milder, Fused-layer CNN accelerators, in *49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2016)* (2016), pp. 1–12
28. Z. Zhao, K.M. Barijough, A. Gerstlauer, DeepThings: distributed adaptive deep learning inference on resource-constrained IoT edge clusters. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **37**(11), 2348–2359 (2018)
29. D. Li, T. Salonidis, N.V. Desai, M.C. Chuah, DeepCham: collaborative edge-mediated adaptive deep learning for mobile object recognition, in *Proceedings of the First ACM/IEEE Symposium on Edge Computing (SEC 2016)* (2016), pp. 64–76
30. S. Yi, Z. Hao, Q. Zhang et al., LAVEA: latency-aware video analytics on edge computing platform, in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing (SEC 2017)* (2017), pp. 1–13
31. X. Xu, Y. Ding, S.X. Hu, M. Niemier, J. Cong, Y. Hu, Y. Shi, Scaling for edge inference of deep neural networks. Nat. Elect. **1**(4), 216–222 (2018)
32. M. Polese, R. Jana, V. Kounev et al., Machine learning at the edge: a data-driven architecture with applications to 5G cellular networks (2018). Preprint. arXiv:1808.07647
33. L. Lai et al., Rethinking machine learning development and deployment for edge devices (2018). Preprint. arXiv:1806.07846
34. P. Meloni, O. Ripolles, D. Solans et al., ALOHA: an architectural-aware framework for deep learning at the edge, in *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications (INTESA 2018)* (2018), pp. 19–26
35. X. Zhang, Y. Wang, S. Lu, L. Liu, L. Xu, W. Shi, OpenEI: an open framework for edge intelligence (2019). Preprint. arXiv:1906.01864
36. J. Zhao, T. Tiplea, R. Mortier, J. Crowcroft, L. Wang, Data analytics service composition and deployment on IoT devices, in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2018)* (2018), pp. 502–504

37. N. Talagala, S. Sundararaman, V. Sridhar, D. Arteaga, Q. Luo, S. Subramanian, S. Ghanta, L. Khermosh, D. Roselli, ECO: harmonizing edge and cloud with ml/dl orchestration, in *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 2018)* (2018)
38. X. Zhang, Y. Wang, W. Shi, pCAMP: performance comparison of machine learning packages on the edges, in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2018)* (2018)
39. A. Ignatov, R. Timofte, W. Chou et al., AI benchmark: running deep neural networks on android smartphones (2018). Preprint. arXiv:1810.01109
40. C. Andrés Ramiro, C. Fiandrino, A. Blanco Pizarro et al., openLEON: an end-to-end emulator from the edge data center to the mobile users carlos, in *Proceedings of the 12th International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization (WiNTECH 2018)* (2018), pp. 19–27
41. Y. Wang, S. Liu, X. Wu, W. Shi, CAVBench: a benchmark suite for connected and autonomous vehicles, in *2018 IEEE/ACM Symposium on Edge Computing (SEC 2018)* (2018), pp. 30–42

Chapter 8
Artificial Intelligence for Optimizing Edge

Abstract DNNs (general DL models) can extract latent data features, while DRL can learn to deal with decision-making problems by interacting with the environment. Computation and storage capabilities of edge nodes, along with the collaboration of the cloud, make it possible to use DL to optimize edge computing networks and systems. With regard to various edge management issues such as edge caching, offloading, communication, security protection, etc., (1) DNNs can process user information and data metrics in the network, as well as perceiving the wireless environment and the status of edge nodes, and based on these information, (2) DRL can be applied to learn the long-term optimal resource management and task scheduling strategies, so as to achieve the intelligent management of the edge, viz. intelligent edge as shown in Tables 8.1, 8.2, and 8.3.

8.1 AI for Adaptive Edge Caching

The rise of various types of intelligent end devices has led to the rapid development of services such as multimedia applications, mobile games, and social networking applications. While this trend has brought increasing traffic pressure to the network architecture, it also shows an interesting feature, that is, the same content is often requested multiple times by devices in the same area. This feature has led researchers to consider the way to cache content to achieve fast response to requests and reduce the traffic load on the network.

From Content Delivery Network (CDN) [1] to caching contents in cellular networks, caching in the network has been investigated over the years to deal with soaring demand for multimedia services [2]. Aligned with the concept of pushing contents near to users, edge caching [3], is deemed as a promising solution for further reducing the redundant data transmission, easing the pressure of cloud data centers, and improving the QoE.

Edge computing deploys a large number of distributed edge nodes at the edge of the network. Edge nodes have the ability to provide services such as storage, transmission, and computing. Edge caching can cache hot content by using edge nodes that are close to the user geographically, so as to achieve rapid response

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_8

Table 8.1 DL for adaptive edge caching

Ref.	DL	Comm. scale	Inputs (states)	Outputs (action)	Loss func. (reward)	Performance
[6]	SDAE	60 users/6 SBSs	User features, content features	Feature-based content popularity matrix	Normalized differences between input features and the consequent reconstruction	QoE improvement: up to 30%; Backhaul offloading: 6.2%
[7]	FCNN	100–200 UEs per cell/7 BSs	Channel conditions, file requests	Caching decisions	Normalized differences between prediction decisions and the optimum	Prediction accuracy: up to 92%; Energy saving: 8% gaps to the optimum
[8]	FCNN	UEs with density 25–30/Multi-tier BSs	Current content popularity, last content placement probability	Content placement probability	Statistical average of the error between the model outputs and the optimal CVX solution	Prediction accuracy: slight degeneration to the optimum
[9]	FCNN CNN	Cars/6 RSUs with MEC servers	Facial images—CNN; Content features—FCNN	Gender and age prediction—CNN; Content request probability—FCNN	N/A—CNN; Cross entropy error—FCNN	Caching accuracy: up to 98.04%
[10]	RNN	20 UEs/10 servers	User historical traces	User location prediction	Cross entropy error	Caching accuracy: up to 75%
[11]	DDPG	Multiple UEs/Single BS	Features of cached contents, current requests	Content replacement	Cache hit rate	Cache hit rate: about 50%

Table 8.2 DL for optimizing edge task offloading

Ref.	DL	Comm. scale	Inputs (states)	Outputs (action)	Loss func. (reward)	Performance
[26]	FCNN	20 miners/Single edge node	Bidder valuation profiles of miners	Assignment probabilities, conditional payments	Expected, negated revenue of the service provider	Revenue increment
[31]	Double-DQL	Single UE	System utilization states, dynamic slack states	DVFS algorithm selection	Average energy consumption	Energy saving: 2–4%
[27]	DQL	Multiple UEs/Single eNodeB	Sum cost of the entire system, available capacity of the MEC server	Offloading decision, resource allocation	Negatively correlated to the sum cost	System cost reduction
[29]	DDPG	Multiple UEs/Single BS with an MEC server	Channel vectors, task queue length	Offloading decision, power allocation	Negative weighted sum of the power consumption and task queue length	Computation cost reduction
[28]	DQL	Single UE/Multiple MEC servers	Previous radio bandwidth, predicted harvested energy, current battery level	MEC server selection, offloading rate	Composition of overall data sharing gains, task drop loss, energy consumption and delay	Energy saving; Delay improvement
[25]	Double-DQL	Single UE/6 BSs with MEC servers	Channel gain states, UE-BS association state, energy queue length, task queue length	Offloading decision, energy units allocation	Composition of task execution delay, task drop times, task queuing delay, task failing penalty and service payment	Offloading performance improvement
[32]	DROO	Multiple UEs/Single MEC server	Channel gain states	Offloading action	Computation rate	Algorithm execution time: less than 0.1 s in 30-UE network

Table 8.3 DL for edge management and maintenance

	Ref.	DL	Comm. scale	Inputs (states)	Outputs (action)	Loss func. (reward)	Performance
Communication	[39]	RNN & LSTM	53 vehicles/20 fog servers	Coordinates of vehicles and interacting fog nodes, time, service cost	Cost prediction	Mean absolute error	Prediction accuracy: 99.2%
	[40]	DQL	4 UEs/Multiple RRHs	Current on-off states of processors, current communication modes of UEs, cache states	Processor state control, communication mode selection	Negative of system energy consumption	System power consumption
Security	[41]	DQL	Multiple UEs/Multiple edge nodes	Jamming power, channel bandwidth, battery levels, user density	Edge node and channel selection, offloading rate, transmit power	Composition of defense costs and secrecy capacity	Signal SINR increasement
Joint optimization	[42]	Double-Dueling DQL	Multiple UEs/5 BSs and 5 MEC servers	Status from each BS, MEC server and content cache	BS allocation, caching decision, offloading decision	Composition of received SNRs, computation capabilities and cache states	System utility increasement
	[43]	AC DRL	20 UEs per router/3 fog nodes	States of requests, fog nodes, tasks, contents, SINR	Decisions about fog node, channel, resource allocation, offloading and caching	Composition of computation offloading delay and content delivery delay	Average service latency: 1.5–4.0 s
	[44]	DQL	50 vehicles/10 RSUs	States of RSUs, vehicles and caches, contact rate, contact times	RSU assignment, caching control and control	Composition of communication, storage and computation cost	Backhaul capacity mitigation; Resource saving

to requests within the service range. Therefore, the edge cache can not only achieve faster request response, but also reduce the repeated transmission of the same content in the network. However, edge caching also faces many challenges. Generally, edge caching needs to address two closely related issues:

1. the content popularity distribution among the coverage of edge nodes is hard to estimate, since it may be different and change with spatiotemporal variation [4];
2. in view of massive heterogeneous devices in edge computing environments, the hierarchical caching architecture and complex network characteristics further perplex the design of content caching strategy [5].

Specifically, the optimal edge caching strategy can only be deduced when the content popularity distribution is known. However, users' predilection for contents is actually unknown since the mobility, personal preference, and connectivity of them may vary all the time. In this section, DL for determining edge caching policies, as illustrated in Fig. 8.1, are discussed.

8.1.1 Use Cases of DNNs

Traditional caching methods are generally with high computational complexity since they require a large number of online optimization iterations to determine (1) the features of users and contents and (2) the strategy of content placement and delivery.

For the first purpose, DL can be used to process raw data collected from the mobile devices of users and hence extract the features of the users and content as a feature-based content popularity matrix. This popularity matrix can quantify the popularity of users and content to provide a numerical basis for caching decisions.

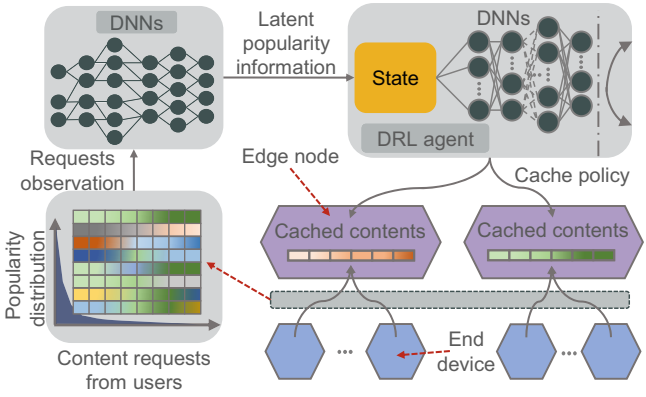

Fig. 8.1 DL and DRL for optimizing the edge caching policy

By this means, the popular content at the core network is estimated by applying feature-based collaborative filtering to the popularity matrix [6].

For the second purpose, when using DNNs to optimize the strategy of edge caching, online heavy computation iterations can be avoided by offline training. A DNN, which consists of an encoder for data regularization and a followed hidden layer, can be trained with solutions generated by optimal or heuristic algorithms and be deployed to determine the cache policy [7], hence avoiding online optimization iterations. Similarly, in [8], inspired by the fact that the output of optimization problem about partial cache refreshing has some patterns, an MLP is trained for accepting the current content popularity and the last content placement probability as input to generate the cache refresh policy.

Although it is possible to design and implement cached content placement and delivery strategies based on DNNs, there are still some shortcomings. As illustrated in [7, 8], the complexity of optimization algorithms can be transferred to the training of DNNs, and thus breaking the practical limitation of employing them. In this case, DL is used to learn input-solution relations, and DNN-based methods are only available when optimization algorithms for the original caching problem exist. Therefore, the performance of DNN-based methods bounds by fixed optimization algorithms and is not self-adapted.

In addition, DL can be utilized for customized edge caching. For example, to minimize content-downloading delay in the self-driving car, an MLP is deployed in the cloud to predict the popularity of contents to be requested, and then the outputs of MLP are delivered to the edge nodes (namely, MEC servers at RSUs in [9]). According to these outputs, each edge node caches contents that are most likely to be requested.

However, for users with different characteristics, their preferences for content are different. Therefore, users can be deeply divided into different categories and then explore the preferences of users in each category, which has a positive impact on improving the hit rate of content caches. On self-driving cars, CNN is chosen to predict the age and gender of the owner. Once these features of owners are identified, k-means clustering [12] and binary classification algorithms are used to determine which contents, already cached in edge nodes, should be further downloaded and cached from edge nodes to the car. Moreover, concerning taking full advantage of users' features, [10] points out that the user's willing to access the content in different environments is varying. Inspired by this, RNN is used to predict the trajectories of users. And based on these predictions, all contents of users' interests are then prefetched and cached in advance at the edge node of each predicted location.

8.1.2 Use Cases of DRL

The function of DNNs described in Sect. 8.1.1 can be deemed as a part of the whole edge caching solution, i.e., the DNN itself does not deal with the whole optimization

problem. Different from these DNNs-based edge caching, DRL can exploit the context of users and networks and take adaptive strategies for maximizing the long-term caching performance [13] as the main body of the optimization method. Traditional RL algorithms are limited by the requirement for handcrafting features and the flaw that hardly handling high-dimensional observation data and actions [14]. Compared to traditional RL irrelevant to DL, such as Q-learning [15] and Multi-Armed Bandit (MAB) learning [4], the advantage of DRL lies in that DNNs can learn key features from the raw observation data. The integrated DRL agent combining RL and DL can optimize its strategies with respect to cache management in edge computing networks directly from high-dimensional observation data.

In [11], DDPG is used to train a DRL agent, in order to maximize the long-term cache hit rate, to make proper cache replacement decisions. This work considers a scenario with a single BS, in which the DRL agent decides whether to cache the requested contents or replace the cached contents. While training the DRL agent, the reward is devised as the cache hit rate. In addition, *Wolpertinger* architecture [16] is utilized to cope with the challenge of large action space. In detail, a primary action set is first set for the DRL agent and then using kNN to map the practical action inputs to one out of this set. In this manner, the action space is narrowed deliberately without missing the optimal caching policy. Compared DQL-based algorithms searching the whole action space, the trained DRL agent with DDPG and *Wolpertinger* architecture is able to achieve competitive cache hit rates while reducing the runtime.

8.2 AI for Optimizing Edge Task Offloading

Edge computing allows edge devices offload part of their computing tasks to the edge node [17], under constraints of energy, delay, computing capability, etc. As shown in Fig. 8.2, these constraints put forward challenges of identifying.

1. Due to the distributed deployment of edge nodes, there will be intersections between different edge nodes in terms of service scope. In this case, an edge device will be in the service range of several edge nodes at the same time. Therefore, it is necessary to consider how the edge device selects a suitable edge node.
2. For edge devices, computing tasks can be executed locally or transmitted to edge nodes for execution. But these two execution methods have their own advantages and disadvantages in terms of resource consumption and latency. Therefore, an efficient and accurate strategy is needed to judge how each task is performed.
3. Due to the variety of application services, there are also a wide variety of tasks that edge devices need to handle. However, different kinds of tasks have different requirements for resources. Therefore, how to efficiently use the existing energy, communication resources, and computing resources has become a challenge.

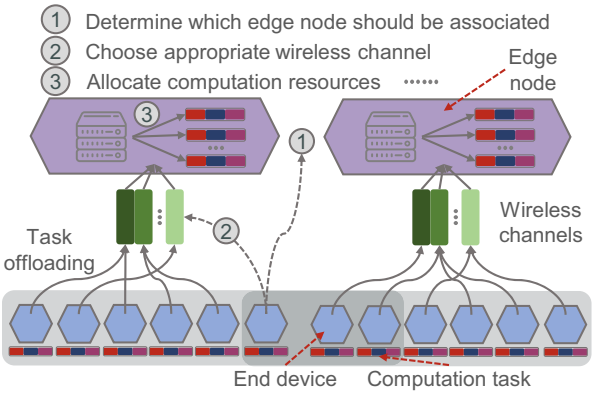

Fig. 8.2 Computation offloading problem in edge computing

To solve this kind of task offloading problem is NP-hard [18], since at least combination optimization of communication and computing resources along with the contention of edge devices is required. Particularly, the optimization should concern both the time-varying wireless environments (such as the varying channel quality) and requests of task offloading, hence drawing the attention of using learning methods [19–29]. Among all these works related to learning-based optimization methods, DL-based approaches have advantages over others when multiple edge nodes and radio channels are available for computation offloading. At this background, large state and action spaces in the whole offloading problem make the conventional learning algorithms [19, 21, 30] infeasible actually.

8.2.1 Use Cases of DNNs

In [23], the computation offloading problem is formulated as a multi-label classification problem. By exhaustively searching the solution in an offline way, the obtained optimal solution can be used to train a DNN with the composite state of the edge computing network as the input, and the offloading decision as the output. Once the DNN is well trained, it can be deployed to receive the observation about the state of edge computing network and then give the optimal offloading decision online. By this means, the exhaustive solution searching may not require to be performed online avoiding belated offloading decision making, and the computation complexity can be transferred to the training process of the DNN. Certainly, using DNNs for approximation may not equal to the performance of exhaustively searching solution, however, the experiment results are also given to determine that this work can approach the optimal solution with a very subtle margin.

Further, a particular offloading scenario with respect to Blockchain is investigated in [26]. The computing and energy resources consumption of mining tasks

on edge devices may limit the practical application of Blockchain in the edge computing network. Naturally, these mining tasks can be offloaded from edge devices to edge nodes, but it may cause unfair edge resource allocation. Thus, all available resources are allocated in the form of auctions to maximize the revenue of the Edge Computing Service Provider (ECSP). Based on an analytical solution of the optimal auction, an MLP can be constructed [26] and trained with valuations of the miners (i.e., edge devices) for maximizing the expected revenue of ECSP.

8.2.2 Use Cases of DRL

Though offloading computation tasks to edge nodes can enhance the processing efficiency of the computation tasks, the reliability of offloading suffers from the potentially low quality of wireless environments. In [22], to maximize offloading utilities, the authors first quantify the influence of various communication modes on the task offloading performance and accordingly propose applying DQL to online select the optimal target edge node and transmission mode. For optimizing the total offloading cost, a DRL agent that modifies Dueling- and Double-DQL [33] can allocate edge computation and bandwidth resources for end devices.

Besides, offloading reliability should also be concerned. Different from other works, [24] considers not only delay violation probability but also the decoding error probability. It points out the coding rate, by which transmitting the data, is crucial to make the offloading meet the required reliability level. In detail, it takes effects of the coding block-length into consideration and formulates a Markov Decision Process (MDP) concerning computational resource allocation in order to improve the average offloading reliability. In this work, DQL is used for finding the optimal strategy of the formulated MDP. First, the DQL agent observes the edge computing environment, consisting of the data size and the waiting time of the tasks to be processed, the queue length of buffered offloaded tasks and the signal-to-noise ratios (SNR) of the downlink. Then, the agent determines the number of CPU cores allocated to the offloaded task on top of the queue. At last, the agent receives a reward or punishment based on the decision it makes. Iteratively, the DQL agent can be trained well and learn the best strategy without knowing the analytical model.

Exploring further on scheduling fine-grained computing resources of the edge device, in [31], Double-DQL [34] is used to determine the best Dynamic Voltage and Frequency Scaling (DVFS) algorithm. Compared to DQL, the experiment results indicate that Double-DQL can save more energy and achieve higher training efficiency. Nonetheless, the action space of DQL-based approaches may increase rapidly with increasing edge devices. Under the circumstances, a pre-classification step can be performed before learning [27] to narrow the action space.

IoT edge environments powered by Energy Harvesting (EH) is investigated in [25, 28]. In EH environments, the energy harvesting makes the offloading problem more complicated, since IoT edge devices can harvest energy from ambient radio-frequency signals. Hence, CNN is used to compress the state space in the learning

process [28]. Further, in [25], inspired by the additive structure of the reward function, Q-function decomposition is applied in Double-DQL, and it improves the vanilla Double-DQL. However, value-based DRL can only deal with discrete action space. To perform more fine-grained power control for local execution and task offloading, policy-gradient-based DRL should be considered. For example, compared to the discrete power control strategy based on DQL, [29] proposes a DRL approach named DDPG with continuous action space, rather discrete action space, to perform more fine-grained power control for local execution and task offloading. In this fashion, the power of edge devices can be adaptively allocated in order to minimizing their long-term average cost, and numerical simulations verify the superiority of the proposed approach over the discrete power control strategy based on DQL.

Freely letting DRL agents take over the whole process of computation offloading may lead to huge computational complexity. Therefore, only employing DNN to make partial decisions can largely reduce the complexity. For instance, in [32], the problem of maximizing the weighted sum computation rate is decomposed into two sub-problems, viz. offloading decision and resource allocation. By only using DRL to deal with the NP-hard offloading decision problem rather than both, the action space of the DRL agent is narrowed, and the offloading performance is not impaired as well since the resource allocation problem is solved optimally.

8.3 AI for Edge Management and Maintenance

Edge AI services are envisioned to be deployed on BSs in cellular networks, as implemented in [35]. Therefore, edge management and maintenance require optimizations from multiple perspectives (including communication perspective). Many works focus on applying AI in wireless communication [36–38]. Nevertheless, management and maintenance at the edge should consider more aspects.

8.3.1 Edge Communication

In edge computing systems, edge nodes usually have more abundant computing resources than edge devices. However, tasks are generated at the edge devices and the results of tasks are also used at the edge devices. In order to use the abundant resources of edge nodes, communication is inevitable. Therefore, it is necessary to ensure the stability, efficiency, and reliability of communication in edge computing systems.

When edge nodes are serving mobile devices (users), mobility issues in edge computing networks should be addressed. DL-based methods can be used to assist the smooth transition of connections between devices and edge nodes. To minimize energy consumption per bit, in [45], the optimal device association strategy is

approximated by a DNN. Meanwhile, a digital twin of network environments is established at the central server for training this DNN offline. To minimize the interruptions of a mobile device moving from an edge node to the next one throughout its moving trajectory, the MLP can be used to predict available edge nodes at a given location and time [39]. Moreover, determining the best edge node, with which the mobile device should associate, still needs to evaluate the cost (the latency of servicing a request) for the interaction between the mobile device and each edge node. Nonetheless, modeling the cost of these interactions requires a more capable learning model. Therefore, a two-layer stacked RNN with LSTM cells is implemented for modeling the cost of interaction. At last, based on the capability of predicting available edge nodes along with corresponding potential cost, the mobile device can associate with the best edge node, and hence the possibility of disruption is minimized.

Aiming at minimizing long-term system power consumption in the communication scenario with multiple modes (to serve various IoT services), i.e., Cloud-Radio Access Networks (C-RAN) mode, Device-to-Device (D2D) mode, and Fog radio Access Point (FAP) mode, DQL can be used to control communication modes of edge devices and on-off states of processors throughout the communicating process [40]. After determining the communication mode and the processors' on-off states of a given edge device, the whole problem can be degraded into a Remote Radio Head (RRH) transmission power minimization problem and solved. Further, TL is integrated with DQL to reduce the required interactions with the environment in the DQL training process while maintaining a similar performance without TL.

8.3.2 Edge Security

Since edge devices generally equipped with limited computation, energy and radio resources, the transmission between them and the edge node is more vulnerable to various attacks, such as jamming attacks and Distributed Denial of Service (DDoS) attacks, compared to cloud computing. Therefore, the security of the edge computing system should be enhanced. First, the system should be able to actively detect unknown attacks, for instance, using AI to extract features of eavesdropping and jamming attacks [46]. According to the attack mode detected, the system determines the strategy of security protection. Certainly, security protection generally requires additional energy consumption and the overhead of both computation and communication. Consequently, each edge device shall optimize its defense strategies, viz. choosing the transmit power, channel, and time, without violating its resource limitation. The optimization is challenging since it is hard to estimate the attack model and the dynamic model of edge computing networks.

DRL-based security solutions can provide secure offloading (from the edge device to the edge node) to against jamming attacks [41] or protect user location privacy and the usage pattern privacy [47]. The edge device observes the status of edge nodes and the attack characteristics and then determines the defense level and key parameters in security protocols. By setting the reward as the anti-jamming communication efficiency, such as the signal-to-interference-plus-noise ratio of the signals, the bit error rate of the received messages, and the protection overhead, the DQL-based security agent can be trained to cope with various types of attacks.

8.3.3 Joint Edge Optimization

Edge computing can cater for the rapid growth of smart devices and the advent of massive computation-intensive and data-consuming applications. Nonetheless, it also makes the operation of future networks even more complex [48]. To manage the complex networks with respect to comprehensive resource optimization [49] is challenging, particularly under the premise of considering key enablers of the future network, including Software-Defined Network (SDN) [50], IoTs, Internet of Vehicles (IoVs).

In general, SDN is designed for separating the control plane from the data plane, and thus allowing the operation over the whole network with a global view. Compared to the distributed nature of edge computing networks, SDN is a centralized approach, and it is challenging to apply SDN to edge computing networks directly. In [51], an SDN-enabled edge computing network catering for smart cities is investigated. To improve the servicing performance of this prototype network, DQL is deployed in its control plane to orchestrate networking, caching, and computing resources.

IoTs is expected to be established for connecting all electronic devices, including sensors, actuators, home appliances, etc. Particularly, edge computing can accommodate the latency requirement of IoT services, and break the capacity limitation of the backhaul link, used to carry data transmission of services. Edge computing can empower IoT systems with more computation-intensive and delay-sensitive services but also raises challenges for efficient management and synergy of storage, computation, and communication resources. For minimizing the average end-to-end servicing delay, policy-gradient-based DRL combined with AC architecture can deal with the assignment of edge nodes, the decision about whether to store the requesting content or not, the choice of the edge node performing the computation tasks and the allocation of computation resources [43].

IoVs is a special case of IoTs and focuses on connected vehicles. Similar to the consideration of integrating networking, caching and computing as in [43], Double-Dueling DQL (i.e., combining Double DQL and Dueling DQL) with more robust performance, can be used to orchestrate available resources to improve the performance of future IoVs [42]. In addition, considering the mobility of vehicles in the IoVs, the hard service deadline constraint might be easily broken, and

this challenge is often either neglected or tackled inadequately because of high complexities. To deal with the mobility challenge, in [44], the mobility of vehicles is first modeled as discrete random jumping, and the time dimension is split into epochs, each of which comprises several time slots. Then, a small timescale DQL model, regarding the granularity of time slot, is devised for incorporating the impact of vehicles' mobility in terms of the carefully designed immediate reward function. At last, a large timescale DQL model is proposed for every time epoch. By using such multi-timescale DRL, issues about both immediate impacts of the mobility and the unbearable large action space in the resource allocation optimization are solved.

8.4 A Practical Case for Adaptive Edge Caching

To further introduce adaptive edge cache, a practical case will be described in detail in this section. In the caching scenario, duplicated download has been a big problem that affects the users' quality of service/experience (QoS/QoE) of current mobile networks. Traditional methods cannot solve this problem well since these methods are lack of the self-adaptive ability in the dynamic environment edge caching and is a promising technology to release the pressure of repeated traffic downloading from the cloud. By virtue of Deep Reinforcement Learning (DRL) and Federated Learning (FL), we can minimize the long-term average content fetching delay of mobile users without requiring any a priori knowledge of content popularity distribution and reduce the privacy risk of users.

8.4.1 Multi-BS Edge Caching Scenario

Assuming there are B BSs in the scene, denoted as $\mathcal{B} = \{b_1, b_2, \ldots, b_B\}$, with the cache size of $C_{\mathcal{B}} = \{c_1^{BS}, c_2^{BS}, \ldots, c_B^{BS}\}$. Mobile users with the local cache buffer c^U are uniformly distributed in the coverage of each BS, represented as $S_B^{BS} = \{U_1^{BS}, U_2^{BS}, \ldots, U_B^{BS}\}$ where U_i^{BS} denotes the set of users. Assuming that there are $\mathcal{F} = \{f_1, f_2, \ldots, f_F\}$ files stored in the content library. For each BS i, it stores local associated users' cache state matrix $\Omega_i = (x_{u,f})^{U_i^{BS} \times F}$, where $x_{u,f} = 1$ means user u has content f, otherwise $x_{u,f} = 0$.

We have content popularity ω_f for each file which follows the Zips distribution and every user also has its preferences p_u^f for each file. By considering the relationship of physical domain and social domain, we can calculate the Device-to-Device (D2D) share probability for every two users. After that we can also get the communication probability between users and BS.

8.4.2 System Formulation

With above definition, we can measure the total D2D sharing in the local BS, we define Ga_b^{D2D} which is calculated via the D2D sharing probability, content size, and user cache state. For a local BS b and all the pairs (u, v) of its associated users U_b^{BS}, v can get the requested content f from u, if u has the content ($x_{u,f} = 1$) and v does not ($x_{v,f} = 0$) with the probability $r_{vu}e_{vu}^c$ during time slot t. Thus the content offload gain via D2D communication between u and v at time slot t can be obtained as $f_{v,t}r_{vu}e_{vu}^c$ where $f_{v,t}$ represents the content size of file that user v requests at time slot t. Then the total D2D gain in the BS b at time slot t can be calculated as:

$$Ga_{b,t}^{D2D} = \sum_{u,v \in U_b^{BS}} f_{v,t}r_{vu}e_{vu}^c x_{u,f}(1 - x_{v,f}). \tag{8.1}$$

It gives us global insight into the D2D sharing activities, which can be regarded as a reference for the system efficiency.

Also if the request cannot be satisfied by the D2D sharing, u can get the content if the content is cached in the local BS. Otherwise, the local BS has to download the content from other BS or the core network, we regard these requests as the cost of the system since these requests increase the burden of the backbone network. We define the backhaul traffic cost at time slot t as $Co_{b,t}^{D2D}$, through the communication probability between user u and BS b, content size, and BS cache state. It can be gotten from

$$Co_{b,t}^{D2D} = \sum_{u \in U_b^{BS}} f_{u,t}P_u^{BS}(1 - x_b^{BS}). \tag{8.2}$$

It gives us insight into the backhaul traffic that cannot be avoided.

Based on the above reference in our system, we define $R_{b,t}(S)$ as the total reference of the system performance at time slot t. The system reward of BS b can be calculated as

$$R_{b,t}(S) = \beta_0 Ga_{b,t}^{D2D} - \beta_1 Co_{b,t}^{D2D}, \tag{8.3}$$

where $\beta_0, \beta_1 \in (0, 1)$. They are users' preferences for D2D communication and BS communication. Note that β_0, β_1 will be different in the different BSs, since every group has their own preference for D2D and BS communication considering the difference in time cost and convenience of these two kinds of communication modes. The system reward $R_{b,t}(S)$ is essential for the following DQN training, it directly decides which content should be replaced in the system.

Once the new requests come, the local BS will determine how to update the cache state of its associated users and which content in the user's cache should be replaced. Therefore the optimization problem can be defined as:

$$\begin{aligned} &\max_{A^*(f^-,f^+)} \sum_{b\in\mathcal{B}} R_{b,t}(S^{'}|S), \\ &s.t. \sum_{i\in\mathcal{F}} x_{u,i} f_i \le c_u^U, \forall u \in U_b^{BS} \\ &\quad \sum_{i\in\mathcal{F}} x_{b,f} f_i \le c_b^{BS} \\ &\quad x_{u,f}, x_{b,f} \in \{0,1\} \end{aligned} \quad , \tag{8.4}$$

where S denotes all the local BS state before the action A^*, and $S^{'}$ denotes the new BS state after action A^*. Our aim is to optimize the best action A^* at current state to maximize the total system reward (maximize the D2D caching gain and minimize the backhaul traffic cost) of the next state while satisfying all the constraints about the cache size of all mobile users and BSs.

8.4.3 Weighted Distributed DQN Training and Cache Replacement

We model the cache replacement problem as a Markov Decision Process (MDP) problem and use a weighted distributed DQN model to solve the problem. The entire training process can be divided into three phases.

1. In the first phase, every BS trains its local DQN Agent based on its local data. Considering the limited computing resources, we set a lightweight DQN model at the edge BS. After a period of time, every local BS will send its parameters and its average system reward to the cloud. The average system reward is mainly used to calculate the weight.
2. In the aggregation phase, each BS disperses its parameters to the cloud server after M epoch and the Cloud will integrate these parameters. Since reward has a strong correlation with the quality of the model, we employ the average reward $\overline{r_b}$ of BS b in a certain time to represent the BS's contribution to the global model.
3. To make the BS agent more adaptive and make full use of the training results of other BSs, we would update the local model via the global model after the aggregation. This can make up for the disadvantages of insufficient computing resources for a single BS.

The computation complexity mainly includes collecting transitions and back propagation. Assume that the length of replay memory is M, the complexity is O(M). Let a and b denote the layer number and the number of units in each layer. The complexity of training parameters with back propagation and gradient descent requires O(mabk), where m is the number of transitions randomly sampled from the

replay memory and k denotes the number of iterations, respectively. Specifically, the replay memory stores M transitions which requires the space complexity of O(M) and it requires the space complexity of O(ab) to deal with the storage complexity by the parameters of DQN.

8.4.4 Conclusion for Edge Caching Case

Basically, this architecture considers the difference of content popularity, user preferences, and social networks in different BSs and establishes a D2D sharing model. It models the cache content replacement problem as a Markov Decision Process (MDP) and trains the DQN model via a reward-based distributed manner which reduces the privacy risk of users, speeds up the training process, and improves the BS's self-adaptive ability in the dynamic environment. Every base station (BS) trains its own DQN model based on its local data and local content popularity. The cloud server will aggregate all the local model parameters in a reward-based adaptive boosting manner and then disperse the global model. Experimental results show that proposed policy achieves better performance compared to the algorithms including FIFO, LRU, LFU, and Centralized DQN.

References

1. M. Hofmann, L. Beaumont, Chapter 3 – caching techniques for web content, in *Content Networking* (2005), pp. 53–79
2. X. Wang, M. Chen, T. Taleb et al., Cache in the air: exploiting content caching and delivery techniques for 5G systems. IEEE Commun. Mag. **52**(2), 131–139 (2014)
3. E. Zeydan, E. Bastug, M. Bennis et al., Big data caching for networking: moving from cloud to edge. IEEE Commun. Mag. **54**(9), 36–42 (2016)
4. J. Song, M. Sheng, T.Q.S. Quek et al., Learning-based content caching and sharing for wireless networks. IEEE Trans. Commun. **65**(10), 4309–4324 (2017)
5. X. Li, X. Wang, P.-J. Wan et al., Hierarchical edge caching in device-to-device aided mobile networks: modeling, optimization, and design. IEEE J. Sel. Areas Commun. **36**(8), 1768–1785 (2018)
6. S. Rathore, J.H. Ryu, P.K. Sharma, J.H. Park, DeepCachNet: a proactive caching framework based on deep learning in cellular networks. IEEE Netw. **33**(3), 130–138 (2019)
7. Z. Chang, L. Lei, Z. Zhou et al., Learn to cache: machine learning for network edge caching in the big data era. IEEE Wirel. Commun. **25**(3), 28–35 (2018)
8. J. Yang, J. Zhang, C. Ma et al., Deep learning-based edge caching for multi-cluster heterogeneous networks, in *Neural Computing and Applications* (2019), pp. 1–12
9. A. Ndikumana, N.H. Tran, C.S. Hong, Deep learning based caching for self-driving car in multi-access edge computing (2018). Preprint. arXiv:1810.01548
10. Y. Tang, K. Guo et al., A smart caching mechanism for mobile multimedia in information centric networking with edge computing. Future Gener. Comput. Syst. **91**, 590–600 (2019)
11. C. Zhong, M.C. Gursoy et al., A deep reinforcement learning-based framework for content caching, in *52nd Annual Conference on Information Sciences and Systems (CISS 2018)* (2018), pp. 1–6

12. T. Kanungo, D. M. Mount et al., An efficient k-means clustering algorithm: analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell. **24**(7), 881–892 (2002)
13. D. Adelman, A.J. Mersereau, Relaxations of weakly coupled stochastic dynamic programs. Oper. Res. **56**(3), 712–727 (2008)
14. H. Zhu, Y. Cao, W. Wang et al., Deep reinforcement learning for mobile edge caching: review, new features, and open issues. IEEE Netw. **32**(6), 50–57 (2018)
15. K. Guo, C. Yang, T. Liu, Caching in base station with recommendation via Q-learning, in *2017 IEEE Wireless Communications and Networking Conference (WCNC 2017)* (2017), pp. 1–6
16. G. Dulac-Arnold, R. Evans, H. van Hasselt et al., Deep reinforcement learning in large discrete action spaces (2015). Preprint. arXiv:1512.07679
17. P. Mach, Z. Becvar, Mobile edge computing: a survey on architecture and computation offloading. IEEE Commun. Surveys Tuts. **19**(3), 1628–1656 (2017). Thirdquarter
18. X. Chen, L. Jiao, W. Li, X. Fu, Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**(5), 2795–2808 (2016)
19. J. Xu, L. Chen et al., Online learning for offloading and autoscaling in energy harvesting mobile edge computing. IEEE Trans. on Cogn. Commun. Netw. **3**(3), 361–373 (2017)
20. T.Q. Dinh, Q.D. La, T.Q.S. Quek, H. Shin, Distributed learning for computation offloading in mobile edge computing. IEEE Trans. Commun. **66**(12), 6353–6367 (2018)
21. T. Chen, G.B. Giannakis, Bandit convex optimization for scalable and dynamic IoT management. IEEE Internet Things J. **6**(1), 1276–1286 (2019)
22. K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, Y. Zhang, Deep learning empowered task offloading for mobile edge computing in urban informatics. IEEE Internet Things J. **6**(5), 7635–7647 (2019)
23. S. Yu, X. Wang, R. Langar, Computation offloading for mobile edge computing: a deep learning approach, in *IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2017)* (2017), pp. 1–6
24. T. Yang, Y. Hu, M.C. Gursoy et al., Deep reinforcement learning based resource allocation in low latency edge computing networks, in *15th International Symposium on Wireless Communication Systems (ISWCS 2018)* (2018), pp. 1–5
25. X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis, Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. IEEE Internet Things J. **6**(3), 4005–4018 (2019)
26. N.C. Luong, Z. Xiong, P. Wang, D. Niyato, Optimal auction for edge computing resource management in mobile blockchain networks: a deep learning approach, in *2018 IEEE International Conference on Communications (ICC 2018)* (2018), pp. 1–6
27. J. Li, H. Gao, T. Lv, Y. Lu, Deep reinforcement learning based computation offloading and resource allocation for MEC, in *2018 IEEE Wireless Communications and Networking Conference (WCNC 2018)* (2018), pp. 1–6
28. M. Min, L. Xiao, Y. Chen et al., Learning-based computation offloading for IoT devices with energy harvesting. IEEE Trans. Veh. Technol. **68**(2), 1930–1941 (2019)
29. Z. Chen, X. Wang, Decentralized computation offloading for multi-user mobile edge computing: a deep reinforcement learning approach (2018). Preprint. arXiv:1812.07394
30. T. Chen et al., Harnessing bandit online learning to low-latency fog computing, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP 2018)* (2018), pp. 6418–6422
31. Q. Zhang, M. Lin, L.T. Yang, Z. Chen, S.U. Khan, P. Li, A double deep q-learning model for energy-efficient edge scheduling. IEEE Trans. Serv. Comput. **12**(05), 739–749 (2019)
32. L. Huang, S. Bi, Y.-j.A. Zhang, Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks (2018). Preprint. arXiv:1808.01977
33. D.C. Nguyen, P.N. Pathirana, M. Ding, A. Seneviratne, Secure computation offloading in blockchain based IoT networks with deep reinforcement learning (2018). Preprint. arXiv:1908.07466

34. H. Van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double Q-learning, in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI 2016)* (2016), pp. 2094–2100
35. C.-Y. Li, H.-Y. Liu et al., Mobile edge computing platform deployment in 4G LTE networks: a middlebox approach, in *{USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 2018)* (2018)
36. Q. Mao, F. Hu, Q. Hao, Deep learning for intelligent wireless networks: a comprehensive survey. IEEE Commun. Surveys Tuts. **20**(4), 2595–2621 (2018). Fourthquarter
37. R. Li, Z. Zhao, X. Zhou et al., Intelligent 5g: when cellular networks meet artificial intelligence. IEEE Wireless Commun. **24**(5), 175–183 (2017)
38. X. Chen, J. Wu, Y. Cai et al., "Energy-efficiency oriented traffic offloading in wireless networks: a brief survey and a learning approach for heterogeneous cellular networks. IEEE J. Sel. Areas Commun. **33**(4), 627–640 (2015)
39. S. Memon et al., Using machine learning for handover optimization in vehicular fog computing, in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (SAC 2019)* (2019), pp. 182–190
40. Y. Sun, M. Peng, S. Mao, Deep reinforcement learning-based mode selection and resource management for green fog radio access networks. IEEE Internet Things J. **6**(2), 1960–1971 (2019)
41. L. Xiao, X. Wan, C. Dai et al., Security in mobile edge caching with reinforcement learning. IEEE Wireless Commun. **25**(3), 116–122 (2018)
42. Y. He, N. Zhao et al., Integrated networking, caching, and computing for connected vehicles: a deep reinforcement learning approach. IEEE Trans. Veh. Technol. **67**(1), 44–55 (2018)
43. Y. Wei, F.R. Yu, M. Song, Z. Han, Joint optimization of caching, computing, and radio resources for fog-enabled IoT using natural actor–critic deep reinforcement learning. IEEE Internet Things J. **6**(2), 2061–2073 (2019)
44. L.T. Tan, R.Q. Hu, Mobility-aware edge caching and computing in vehicle networks: a deep reinforcement learning. IEEE Trans. Veh. Technol. **67**(11), 10190–10203 (2018)
45. R. Dong, C. She, W. Hardjawana, Y. Li, B. Vucetic, Deep learning for hybrid 5G services in mobile edge computing systems: learn from a digital twin. IEEE Trans. Wirel. Commun. **18**(10), 4692–4707 (2019)
46. Y. Chen, Y. Zhang, S. Maharjan, M. Alam, T. Wu, Deep learning for secure mobile edge computing in cyber-physical transportation systems. IEEE Netw. **33**(4), 36–41 (2019)
47. M. Min, X. Wan, L. Xiao et al., Learning-based privacy-aware offloading for healthcare IoT with energy harvesting. IEEE Internet Things J. **6**, 4307–4316 (2018)
48. T.E. Bogale, X. Wang, L.B. Le, Machine intelligence techniques for next-generation context-aware wireless networks (2018). Preprint. arXiv:1801.04223
49. S. Wang, X. Zhang, Y. Zhang et al., A survey on mobile edge networks: convergence of computing, caching and communications. IEEE Access **5**, 6757–6779 (2017)
50. D. Kreutz et al., Software-defined networking: a comprehensive survey. Proc. IEEE **103**(1), 14–76 (2015)
51. Y. He, F.R. Yu, N. Zhao et al., Software-defined networks with mobile edge computing and caching for smart cities: a big data deep reinforcement learning approach. IEEE Commun. Mag. **55**(12), 31–37 (2017)

Part III
Challenges and Conclusions

Chapter 9
Lessons Learned and Open Challenges

Abstract To identify existing challenges and circumvent potential misleading directions, we briefly introduce the potential scenario of "AI Application on Edge," and separately discuss open issues related to four enabling technologies that we focus on, i.e., "AI Inference in Edge," "Edge Computing for AI," "AI Training at Edge," and "AI for Optimizing Edge."

9.1 More Promising Applications

If AI and edge are well-integrated, they can offer great potential for the development of innovative applications. There are still many areas to be explored to provide operators, suppliers, and third parties with new business opportunities and revenue streams.

For example, with more AI techniques are universally embedded in these emerged applications, the introduced processing delay and additional computation cost make the cloud gaming architecture struggle to meet the latency requirements. Edge computing architectures, near to users, can be leveraged with the cloud to form a hybrid gaming architecture. Besides, intelligent driving involves speech recognition, image recognition, intelligent decision making, etc. Various AI applications in intelligent driving, such as collision warning, require edge computing platforms to ensure millisecond-level interaction delay. In addition, edge perception is more conducive to analyze the traffic environment around the vehicle, thus enhancing driving safety.

In addition to society and life, edge computing also has great application prospects in the military field, one of which is vision sharing. When the armed perform anti-terrorism operations or special tasks, due to obstacles and their limited vision, the scope they can see is limited, which also increases the danger and unknownness. The early solution was to implement information exchange between soldiers on the battlefield based on Wireless sensor networks (WSN) [1], but this method still has great uncertainty for soldiers. Edge computing and deep learning can solve this problem very well. By wearing AR glasses with cameras for each combatant, the edge device can collect and combine the visual field information

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_9

from all personnel, collect other image information from drones, and use deep learning algorithms to display all important information in the entire scene in the AR glasses of each combatant. And it improves the efficiency and safety of combat in mission execution.

9.2 General AI Model for Inference

Edge devices often have resource limitations and the AI model will cause a lot of resource consumption and delay during inference. Therefore, when deploying AI in edge devices, it is necessary to accelerate AI inference by model optimization. In this section, lessons learned and future directions for "AI inference in edge," with respect to model compression, model segmentation, and EEoI, used to optimize AI models, are discussed.

9.2.1 Ambiguous Performance Metrics

For an Edge AI service for a specific task, there are usually a series of AI model candidates that can accomplish the task. However, it is difficult for service providers to choose the right AI model for each service. Due to the uncertain characteristics of edge computing networks (varying wireless channel qualities, unpredictable concurrent service requests, etc.), commonly used standard performance indicators (such as top-k accuracy [2] or mean average accuracy [3]) cannot reflect the runtime performance of AI model inference in the edge. Therefore, this will lead to the inability to accurately quantify and compare the performance of different AI models, which will not only cause confusion when selecting the appropriate AI model but also restrict the optimization and improvement of the AI model.

For Edge AI services, besides model accuracy, inference delay, resource consumption, and service revenue are also key indicators. Edge AI services often not only focus on the accuracy of the model. Due to the different types of services and application scenarios, an edge AI service is often related to multiple indicators. For instance, in [4], the author needs to comprehensively consider the complex interaction of multiple indicators such as model accuracy, video quality, battery constraints, network data usage, and network conditions in the object detection service to determine the best offloading strategy. Similarly, in [5], the author should comprehensively consider network traffic and running time in the handwriting recognition scenario to optimize performance. When multiple indicators are involved in a service, a new problem arises, namely the trade-off between multiple indicators. Due to the characteristics of edge AI services, different indicators have different effects on services. However, how to accurately balance the multiple indicators to maximize the comprehensive performance has become an urgent problem. Therefore, we need to identify the key performance indicators of

Edge AI, quantitatively analyze the factors affecting them, and explore the trade-offs between these indicators to help improve the efficiency of Edge AI deployment.

9.2.2 Generalization of EEoI

Currently, EEoI can be applied to classification problems in DL [6], but there is no generalized solution for a wider range of applications. Therefore, we need to further improve the combination of EEoI and DL so that it can be applied in a wider range of fields.

Furthermore, in order to build an intelligent edge and support edge intelligence, not only DL but also the possibility of applying EEoI to DRL should be explored, since applying DRL to real-time resource management for the edge, as discussed in Sect. 8, requires stringent response speed. EEoI can simplify the reasoning process and speed up the response speed, which can well meet this requirement. Therefore, the application of EEoI in DRL is of great research significance, but there is currently no practical research in this area.

9.2.3 Hybrid Model Modification

At present, many researchers have proposed optimization methods for AI models. However, these optimization methods are relatively independent and different characteristics, so these methods have the potential for compatibility. Coordination issues with respect to model optimization, model segmentation, and EEoI should be thought over. These customized AI models are often used independently to enable "end–edge–cloud" collaboration. However, in order to optimize the performance, the application of hybrid optimization methods can be considered. In the hybrid application of multiple optimization methods, it may be possible to break through the bottleneck of single optimization method on performance improvement.

In the hybrid use of multiple optimization methods, the characteristics of various optimization methods also need to be considered. Model optimizations, such as model quantification and pruning, may be required on the end and edge sides, but because of the sufficient computation resources, the cloud does not need to take the risk of model accuracy to use these optimizations. In addition, the possible correlations and effects between different optimizations should be considered to avoid interference with each other and affect the overall performance.

Therefore, how to design a hybrid precision scheme, that is, to effectively combine the simplified AI models in the edge with the raw AI model in the cloud is important.

9.2.4 Coordination Between AI Training and Inference

Pruning, quantizing, and introducing EEoI into trained raw AI models require retraining to give them the desired inference performance. However, during the training of the AI model, it will result in resource consumption and additional delay. Therefore, when applying the optimization method, not only the performance improvement should be considered but also the additional cost it causes.

In general, customized models can be trained offline in the cloud. However, the advantage of edge computing lies in its response speed and might be neutralized because of belated AI training. Therefore, the relationship between training and inference must be fully weighed to avoid spending more resources on training in pursuit of small efficiency improvements. This approach may lead to negative optimization, that is, the performance of the AI model after optimization is not as good as that of the original AI model.

Moreover, due to a large number of heterogeneous devices in the edge and the dynamic network environment, the customization requirements of AI models are not monotonous. Then, is this continuous model training requirement reasonable, and will it affect the timeliness of model inference? How to design a mechanism to avoid these side effects?

9.3 Complete Edge Architecture for AI

Edge intelligence and intelligent edge require a complete system framework, covering data acquisition, service deployment, and task processing. In this section, we discuss challenges for "Edge computing for AI" to build a complete edge computing framework for AI, as summarized in Table 9.1.

9.3.1 Edge for Data Processing

Both pervasively deployed AI services on the edge and AI algorithms for optimizing edge cannot be realized without data acquiring. Edge architecture should be able to efficiently acquire and process the original data, sensed or collected by edge devices, and then feed them to AI models. Hence, in order to support AI inference and training, practical edge solutions for data processing are required.

In smart surveillance, adaptively acquiring video streams at edge and then transmitting them to cloud is a natural way to alleviate the workload of edge devices and reduce the potential communication cost. However, these video sources with different qualities may harm the video analysis confidence. Therefore, how to abstract the effective information from the source while ensuring the analysis performance becomes a challenge need to be considered. For this adaptive data

Table 9.1 Research directions and challenges of "Edge Computing for AI"

Research direction	Expected objective	Challenge or potential solution	Metric
Edge for data processing	Efficiently acquire and process massive sensed or collected data by edge infrastructures	Adaptive data acquirement	Overhead
		Adaptive data compression	Processing speed
		Universal or customized architecture	Throughput
Reduction of DL computation	Dispose DL requests at the edge without redundant computation	DRL support	Overhead
		Cache inference results	Inference latency
		Deploy multiple DL models	Model accuracy
		Sharing intermediate DL inference results	
Microservice for AI services	Host AI services on microservice frameworks deployed on the edge	Microservice framework for edge computing	Overhead
		Live migration of microservices	Throughput
		Coordination between the cloud and the edge	Stability
Incentive and trusty offloading	Provide edge nodes with incentives to take over AI computations	Apply blockchain in the edge architecture	Overhead
	Guarantee the security to avoid the risks from anonymous edge nodes	Existing blockchain solutions do not support the execution of DL	Throughput
			Security
			Inference latency
Edge architecture and testbed for AI	Automate phases of the AI development process	Exploit underlying edge hardware in terms of performance and power	Overhead
	Practically supporting ideas in Sect. 7.3	Develop a standard and comprehensive testbed	Inference latency
	Testify and optimize the performance of AI services on the edge		Throughput
			Stability

acquirement, with respect to vision applications, RoI (Region-of-Interest) of a frame out of video streams can be designated and transmitted with higher quality, and, thus, achieving high analysis accuracy while conserving transmission bandwidth and energy [7].

In addition, it is a better choice to compress the transmitted data. On the one hand, it can alleviate the bandwidth pressure of the network. On the other hand, data transmission delay can be reduced to provide a better service quality. Designing an adaptive data compression mechanism, aiming to find a balance between compression ratio and transmission delay, is feasible in vision applications [8]. Most existed works focus only on the AI services related to videos or images. However, there are a wide variety of AI services, for which the structures and characteristics of data are heterogeneous, need to be concerned. Developing a more universal or customized architecture of edge data processing, such as adaptive data acquirement and data compression, for different AI services will be helpful.

9.3.2 Microservice for Edge AI Services

Edge and cloud services have recently started undergoing a major shift from monolithic entities to graphs of hundreds of loosely coupled microservices [9]. Executing AI computations may need a series of software dependencies, and it calls for a solution for isolating different AI services on the shared resources.

At present, the microservice framework, deployed on the edge for hosting AI services, is in its infant [10], due to several critical challenges: (1) Handling AI deployment and management flexibly; (2) Achieving live migration of microservices to reduce migration times and unavailability of AI services due to user mobilities; (3) Orchestrating resources among the cloud and distributed edge infrastructures to achieve better performance, as illustrated in Sect. 7.3.3.

9.3.3 Incentive and Trusty Offloading Mechanism for AI

Heavy AI computations on resource-limited end devices can be offloaded to nearby edge nodes (Sect. 7.3). However, there are still several issues, (1) an incentive mechanism should be established for stimulating edge nodes to take over AI computations; (2) the security should be guaranteed to avoid the risks from anonymous edge nodes [11].

Blockchain, as a decentralized public database storing transaction records across participated devices, can avoid the risk of tampering the records [12]. By taking advantage of these characteristics, incentive and trust problems with respect to computation offloading can potentially be tackled. To be specific, all end devices and edge nodes have to first put down deposits to the blockchain to participate. The end device request the help of edge nodes for AI computation, and meantime send a

"require" transaction to the blockchain with a bounty. Once an edge nodes complete the computation, it returns results to the end device with sending a "complete" transaction to the blockchain. After a while, other participated edge nodes also execute the offloaded task and validate the former recorded result. At last, for incentives, firstly recorded edge nodes win the game and be awarded [13]. However, this idea about blockchained edge is still in its infancy. Existing blockchains such as Ethereum [14] do not support the execution of complex AI computations, which raises the challenge of adjusting blockchain structure and protocol in order to break this limitation.

9.3.4 Integration with "AI for Optimizing Edge"

End devices, edge nodes, and base stations in edge computing networks are expected to run various AI models and deploy corresponding services in the future. In order to make full use of decentralized resources of edge computing, and to establish connections with existing cloud computing infrastructure, dividing the computation-intensive AI model into sub-tasks and effectively offloading these tasks between edge devices for collaboration are essential. Owing to deployment environments of Edge AI are usually highly dynamic, edge computing frameworks need excellent online resource orchestration and parameter configuration to support a large number of AI services. Heterogeneous computation resources, real-time joint optimization of communication and cache resources, and high-dimensional system parameter configuration are critical. We have introduced various theoretical methods to optimize edge computing frameworks (networks) with DL technologies in Sect. 8. Nonetheless, there is currently no relevant work to deeply study the performance analysis of deploying and using these DL technologies for long-term online resource orchestration in practical edge computing networks or testbeds. We believe that "Edge computing for AI" should continue to focus on how to integrate "AI for Optimizing Edge" into the edge computing framework to realize the above vision.

9.4 Practical Training Principles at Edge

Compared with AI inference in the edge, AI training at the edge is currently mainly limited by the weak performance of edge devices and the fact that most Edge AI frameworks or libraries still do not support training. At present, most studies are at the theoretical level, i.e., simulating the process of AI training at the edge. In this section, we point out the lessons learned and challenges in "AI Training at Edge."

9.4.1 Data Parallelism Versus Model Parallelism

AI models are both computation and memory intensive. When they become deeper and larger, it is not feasible to acquire their inference results or train them well by a single device. Therefore, large AI models are trained in distributed manners over thousands of CPU or GPU cores, in terms of data parallelism, model parallelism, or their combination (Sect. 3.4). However, differing from parallel training over bus-connected or switch-connected CPUs or GPUs in the cloud, perform model training at distributed edge devices should further consider wireless environments, device configurations, privacies, etc.

At present, FL only copies the whole AI model to every participated edge devices, namely in the manner of data parallelism. Hence, taking the limited computing capabilities of edge devices (at least for now) into consideration, partitioning a large-scale AI model and allocating these segments to different edge devices for training may be a more feasible and practical solution. Certainly, this does not mean abandoning the native data parallelism of FL, instead, posing the challenge of blending data parallelism and model parallelism particularly for training AI models at the edge, as illustrated in Fig. 9.1.

9.4.2 Training Data Resources

Currently, most of the AI training frameworks at the edge are aimed at supervised learning tasks, and test their performance with complete datasets. However, in practical scenarios, we cannot assume that all data in the edge computing network are labeled and with a correctness guarantee. For unsupervised learning tasks such as DRL, we certainly do not need to pay too much attention to the production of training data. For example, the training data required for DRL composed of the observed state vectors and rewards obtained by interacting with the environment. These training data can generate automatically when the system is running. But for a wider range of supervised learning tasks, how edge nodes and devices find the exact training data for model training? The application of vanilla FL is using RNN for next-word-prediction [15], in which the training data can be obtained along with

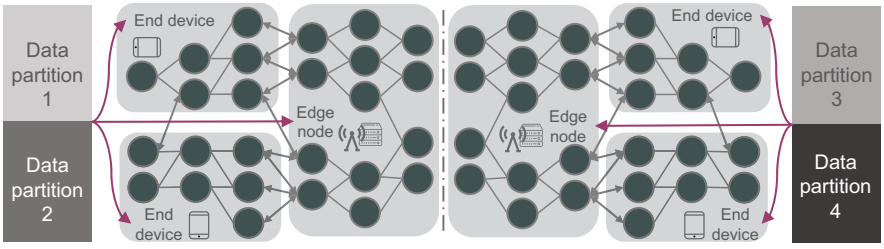

Fig. 9.1 AI training at the edge by both data and model parallelism

users' daily inputs. Nonetheless, for extensive Edge AI services concerning video analysis, where are their training data from. If all training data is manually labeled and uploaded to the cloud data center, and then distributed to edge devices by the cloud, the original intention of FL is obviously violated. One possible solution is to enable edge devices to construct their labeled data by learning "labeled data" from each other. We believe that the production of training data and the application scenarios of AI models training at the edge should first be clarified in the future, and the necessity and feasibility of AI model training at the edge should be discussed as well.

9.4.3 Asynchronous FL at Edge

Existing FL methods [15, 16] focus on synchronous training, and can only process hundreds of devices in parallel. However, this synchronous updating mode potentially cannot scale well, and is inefficient and inflexible in view of two key properties of FL, specifically, (1) infrequent training tasks, since edge devices typically have weaker computing power and limited battery endurance and thus cannot afford intensive training tasks; (2) limited and uncertain communication between edge devices, compared to typical distributed training in the cloud.

Thus, whenever the global model is updating, the server is limited to selecting from a subset of available edge devices to trigger a training task. In addition, due to limited computing power and battery endurance, task scheduling varies from device to device, making it difficult to synchronize selected devices at the end of each epoch. Some devices may no longer be available when they should be synchronized, and hence the server must determine the timeout threshold to discard the laggard. If the number of surviving devices is too small, the server has to discard the entire epoch including all received updates. These bottlenecks in FL can potentially be addressed by asynchronous training mechanisms [17–19]. Adequately selecting clients in each training period with resource constraints may also help. By setting a certain deadline for clients to download, update, and upload AI models, the central server can determine which clients to perform local training such that it can aggregate as many client updates as possible in each period, thus allowing the server to accelerate performance improvement in AI models [20].

9.4.4 Transfer Learning-Based Training

Due to resource constraints, training and deploying computation-intensive AI models on edge devices such as mobile phones is challenging. In order to facilitate learning on such resource-constrained edge devices, TL can be utilized. For instance, in order to reduce the amount of training data and speeding up the training process, using unlabeled data to transfer knowledge between edge devices can be

adopted [21]. By using the cross-modal transfer in the learning of edge devices across different sensing modalities, required labeled data and the training process can be largely reduced and accelerated, respectively.

Besides, KD, as a method of TL, can also be exploited thanks to several advantages [22]: (1) using information from well-trained large AI models (teachers) to help lightweight AI models (students), expected to be deployed on edge devices, converge faster; (2) improving the accuracy of students; (3) helping students become more general instead of being overfitted by a certain set of data. Although results of [21, 22] show some prospects, further research is needed to extend the TL-based training method to AI applications with different types of perceptual data.

9.5 Deployment and Improvement of Intelligent Edge

There have been many attempts to use DL to optimize and schedule resources in edge computing networks. In this regard, there are many potential areas where DL can be applied, including online content streaming [23], routing and traffic control [24, 25], etc. However, since DL solutions do not rely entirely on accurate modeling of networks and devices, finding a scenario where DL can be applied is not the most important concern. Besides, if applying DL to optimize real-time edge computing networks, the training and inference of AI models or DRL algorithms may bring certain side effects, such as the additional bandwidth consumed by training data transmission and the latency of AI inference.

Existing works mainly concern about solutions of "AI for Optimizing Edge" at the high level, but overlook the practical feasibility at the low level. Though DL exhibits its theoretical performance, the deployment issues of DNNs/DRL should be carefully considered (as illustrated in Fig. 9.2):

- Where DL and DRL should be deployed, in view of the resource overhead of them and the requirement of managing edge computing networks in real time?
- When using DL to determine caching policies or optimize task offloading, will the benefits of DL be neutralized by the bandwidth consumption and the processing delay brought by DL itself?

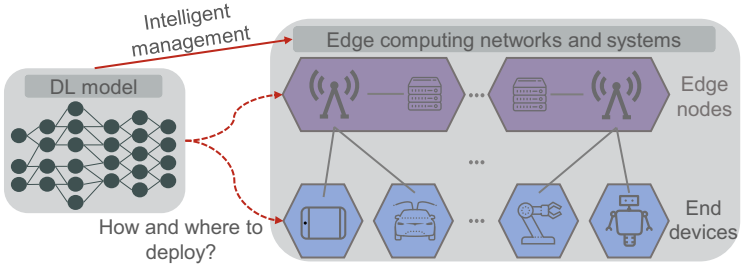

Fig. 9.2 Deployment issues of intelligent edge, i.e., how and where to deploy AI models for optimizing edge computing networks (systems)

- How to explore and improve edge computing architectures in Sect. 7 to support "AI for Optimizing Edge?"
- Are the ideas of customized AI models, introduced in Sect. 5, can help to facilitate the practical deployment?
- How to modify the training principles in Sect. 6 to enhance the performance of DL training, in order to meet the timeliness of edge management?

Besides, the abilities of the state-of-the-art DL or DRL, such as Multi-Agent Deep Reinforcement Learning [26–28], Graph Neural Networks (GNNs) [29, 30], can also be exploited to facilitate this process. For example, end devices, edge nodes, and the cloud can be deemed as individual agents. By this means, each agent trains its own strategy according to its local imperfect observations, and all participated agents work together for optimizing edge computing networks. In addition, the structure of edge computing networks across the end, the edge, and the cloud is actually an immense graph, which comprises massive latent structure information, e.g., the connection and bandwidth between devices. For better understanding edge computing networks, GNNs, which focuses on extracting features from graph structures instead of two-dimensional meshes and one-dimensional sequences, might be a promising method.

References

1. M.P. Đurišić, Z. Tafa, G. Dimić, et al., A survey of military applications of wireless sensor networks, in *2012 Mediterranean Conference on Embedded Computing (MECO)* (2012), pp. 196–199
2. J. Jiang, G. Ananthanarayanan, P. Bodik, S. Sen, I. Stoica, Chameleon: scalable adaptation of video analytics, in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2018)* (2018), pp. 253–266
3. L.N. Huynh, Y. Lee, R.K. Balan, DeepMon: mobile GPU-based deep learning framework for continuous vision applications, in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys 2017)* (2017), pp. 82–95
4. X. Ran, H. Chen, X. Zhu, Z. Liu, J. Chen, DeepDecision: a mobile deep learning framework for edge video analytics, in *2018 IEEE Conference on Computer Communications (INFOCOM 2018)* (2018), pp. 1421–1429
5. Y. Huang, X. Ma, X. Fan et al., When deep learning meets edge computing, in *IEEE 25th International Conference on Network Protocols (ICNP 2017)* (2017), pp. 1–2
6. S. Teerapittayanon et al., BranchyNet: fast inference via early exiting from deep neural networks, in *Proceedings of the 23rd International Conference on Pattern Recognition (ICPR 2016)* (2016), pp. 2464–2469
7. B.A. Mudassar, J.H. Ko, S. Mukhopadhyay, Edge-cloud collaborative processing for intelligent internet of things, in *Proceedings of the 55th Annual Design Automation Conference (DAC 2018)* (2018), pp. 1–6
8. J. Ren, Y. Guo, D. Zhang et al., Distributed and efficient object detection in edge computing: challenges and solutions. IEEE Netw. **32**(6), 137–143 (2018)
9. Y. Gan, Y. Zhang, D. Cheng et al., An open-source benchmark suite for microservices and their hardware-software implications for cloud and edge systems, in *Proceedings of the Twenty Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2019)* (2019)

10. M. Alam, J. Rufino, J. Ferreira, S.H. Ahmed, N. Shah, Y. Chen, Orchestration of microservices for IoT using docker and edge computing. IEEE Commun. Mag. **56**(9), 118–123 (2018)
11. J. Xu, S. Wang, B. Bhargava, F. Yang, A blockchain-enabled trustless crowd-intelligence ecosystem on mobile edge computing. IEEE Trans. Ind. Inf. **15**, 3538–3547 (2019)
12. Z. Zheng, S. Xie, H. Dai et al., An overview of blockchain technology: architecture, consensus, and future trends, in *2017 IEEE International Congress on Big Data (BigData Congress 2017)* (2017), pp. 557–564
13. J.-y. Kim, S.-M. Moon, Blockchain-based edge computing for deep neural network applications, in *Proceedings of the Workshop on INTelligent Embedded Systems Architectures and Applications (INTESA 2018)* (2018), pp. 53–55
14. G. Wood, Ethereum: a secure decentralised generalised transaction ledger (2014) [Online]. Available: http://gavwood.com/Paper.pdf
15. K. Bonawitz, H. Eichner et al., Towards federated learning at scale: system design (2019). Preprint. arXiv:1902.01046
16. H.B. McMahan, E. Moore, D. Ramage et al., Communication-efficient learning of deep networks from decentralized data, in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS 2017)* (2017), pp. 1273–1282
17. S. Zheng, Q. Meng, T. Wang et al., Asynchronous stochastic gradient descent with delay compensation, in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)* (2017), pp. 4120–4129
18. C. Xie, S. Koyejo, I. Gupta, Asynchronous federated optimization (2019). Preprint. arXiv:1903.03934
19. W. Wu, L. He, W. Lin, RuiMao, S. Jarvis, SAFA: a semi-asynchronous protocol for fast federated learning with low overhead (2019). Preprint. arXiv:1910.01355
20. T. Nishio, R. Yonetani, Client selection for federated learning with heterogeneous resources in mobile edge (2018). Preprint. arXiv:1804.08333
21. T. Xing, S.S. Sandha, B. Balaji et al., Enabling edge devices that learn from each other: cross modal training for activity recognition, in *Proceedings of the 1st International Workshop on Edge Systems, Analytics and Networking (EdgeSys 2018)* (2018), pp. 37–42
22. R. Sharma, S. Biookaghazadeh et al., Are existing knowledge transfer techniques effective for deep learning on edge devices? in *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC 2018)* (2018), pp. 15–16
23. J. Yoon, P. Liu, S. Banerjee, Low-cost video transcoding at the wireless edge, in *2016 IEEE/ACM Symposium on Edge Computing (SEC 2016)* (2016), pp. 129–141
24. N. Kato et al., The deep learning vision for heterogeneous network traffic control: proposal, challenges, and future perspective. IEEE Wireless Commun. **24**(3), 146–153 (2017)
25. Z.M. Fadlullah, F. Tang, B. Mao et al., State-of-the-art deep learning: evolving machine intelligence toward tomorrow's intelligent network traffic control systems. IEEE Commun. Surveys Tuts. **19**(4), 2432–2455 (2017). Fourthquarter
26. J. Foerster, I.A. Assael et al., Learning to communicate with deep multi-agent reinforcement learning, in *Advances in Neural Information Processing Systems 29 (NeurIPS 2016)* (2016), pp. 2137–2145
27. S. Omidshafiei, J. Pazis, C. Amato et al., Deep decentralized multi-task multi-agent reinforcement learning under partial observability, in *Proceedings of the 34th International Conference on Machine Learning (ICML 2017)* (2017), pp. 2681–2690
28. R. Lowe, Y. WU et al., Multi-agent actor-critic for mixed cooperative-competitive environments, in *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)* (2017), pp. 6379–6390
29. J. Zhou, G. Cui et al., Graph neural networks: a review of methods and applications (2018). Preprint. arXiv:1812.08434
30. Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: a survey (2018). Preprint. arXiv:1812.04202

Chapter 10
Conclusions

Artificial intelligence and edge computing are expected to benefit each other. This book has comprehensively introduced and discussed various applicable scenarios and fundamental enabling techniques for *edge intelligence* and *intelligent edge*. In summary, the key issue of extending AI from the cloud to the edge of the network is: under the multiple constraints of networking, communication, computing power, and energy consumption, how to devise and develop edge computing architecture to achieve the best performance of AI training and inference. As the computing power of the edge increases, edge intelligence will become common, and intelligent edge will play an important supporting role to improve the performance of edge intelligence. We hope that this survey will increase discussions and research efforts on AI/edge integration that will advance future edge AI applications and services.

© The Editor(s) (if applicable) and The Author(s), under exclusive licence to Springer Nature Singapore Pte Ltd. 2020

X. Wang et al., *Edge AI*, https://doi.org/10.1007/978-981-15-6186-3_10